Západočeská univerzita v Plzni

Fakulta aplikovaných věd

Katedra informatiky a výpočetní techniky

Diplomová práce

Zvýšení bezpečnosti Linuxu

Plzeň 2017 Miroslav Marek

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 28. června 2017

Miroslav Marek

Poděkování

Děkuji Ing. Michalovi Švambergovi za odborné vedení a čas věnovaný mě při tvorbě tohoto dokumentu. Mé poděkováni patří též CIV za poskytnutí infrastruktury pro účely měření.

Abstrakt

Diplomová práce Zvýšení bezpečnosti Linuxu je dokument, jehož cílem je seznámení čtenáře s klasifikací bezpečnosti informačních systémů TCSEC, s úrovní bezpečnosti operačních systémů GNU Linux, s mechanismem správy paměti, kterého se týká největší díl zranitelností a na jehož vylepšení je zvýšení bezpečnosti z větší části postaveno, s vybranými zranitelnostmi zapříčiněnými nedokonalostí Linuxu, s hrozbami, které mohou těchto zranitelností zneužít k úmyslnému poškození systému, s vybranými řešeními, které zranitelnosti buďto přímo odstraní nebo alespoň minimalizují rozsah škod, který by mohly způsobit případné hrozby, s analýzou jednotlivých komponent těchto řešení, s jejich dopadem na výkonnost zabezpečovaného systému, se zhodnocením tohoto jejich dopadu a s doporučením pro jejich nasazení pro různá prostředí.

Abstract

The diploma thesis Increase Linux security is a document, whose aim is introduce the reader with the security classification of information systems TCSEC, with the security level of operating systems GNU Linux, with the memory management mechanism that hosts the largest share of vulnerabilities and on which the increased security is largely built, with selected vulnerabilities caused by Linux's imperfections, with threats that can exploit these vulnerabilities to deliberate damage to the system, with selected solutions that will either eliminate the vulnerabilities directly or at least minimize the extent of damage that could be caused by potential threats, with the analysis of the individual components of these solutions, with their impact on the performance of the secured system, with an assessment of this impact and with recommendations for their deployment for different environments.

Obsah

1. Úvod

A k čemu bude Zvýšení bezpečnosti Linuxu dobré? To záleží na cennosti dat, se kterými systém, v němž se „bezpečnější“ jádro nasadí, pracuje, či se kterými pracují systémy, jimž má zabezpečovaný systém poskytovat ochranu[1]. **Bezpečnostní mechanismy dnes přítomné ve vanilla Linuxu či nějaké jeho distribuční derivaci pro většinu nasazení stačí**. Za naprostý základ mezi bezpečnostními mechanismy lze považovat např. DAC v podobě standardních unixových práv a POSIX ACL, capabilities, segmentaci či konečně základ všeho zabezpečení v IT, bez kterého by ostatní bezpečnostní mechanismy nemohly existovat – Rings.

Tempo přírůstku bezpečnostních mechanismů ale rozhodně neslábne[2]. **Mnohé** z nich však **bývají** ve výchozím nastavení jádra **deaktivované či se do jádra vůbec nedostanou** a zájemci o ně tak musí vynaložit nikoliv nepatrné úsilí, chtějí-li je na svých systémech zprovoznit. **Takové** bezpečnostní mechanismy **lze nazvat nadstandardními**. Už jen skutečnost, že vznikají, dává tušit, že Zvýšení bezpečnosti Linuxu by k něčemu dobré být mohlo.

Čím jsou data, s nimiž systém zachází nebo kterým poskytuje ochranu, cennější, tím větší je motivace mnohých osob se těchto dat zmocnit či je nějak modifikovat ve svůj prospěch. Problémem na tom všem je, že to dělají, přestože k tomu nemají pověření. Proto jádro disponuje již v základu celou řadou mechanismů, jejichž účelem je vynucení přístupových oprávnění a v konečném důsledku tedy i zamezení takovému jednání. Lačnost po datech ze strany těchto osob bez pověření však standardní bezpečnostní mechanismy často překoná čili přesněji řečeno motivuje k jejich „obejití jinou cestou“. Provozovatelé systémů si zpravidla nepřejí, aby se jim kdokoliv bez pověření prohánět po systému a proto tvoří poptávku po nadstandardních bezpečnostních mechanismech. Pro ty tak existuje trh a bezpečnostní odborníci jej plní svými vynálezy, které následně provozovatelé systémů nasazují. Ke smůle provozovatelů však motivace a zejména znalosti osob bez pověření rostou společně s tím, jak jsou systémy postupně více a více zabezpečovány.

Nadstandardní bezpečnostní mechanismy přestávají býti nadstandardními v okamžiku, kdy vzniknou nástroje, které automatizují exploitování zranitelností, které tyto nadstandardní bezpečnostní mechanismy zneškodňují a kdy se zároveň tyto zranitelnosti stanou mediálně známými. Mediální podpora zranitelnostem a vznik nástrojů automatizujících jejich exploitování totiž vedou k prudkému poklesu požadavků na znalosti, které jsou k jejich exploitování jinak běžně potřebné. Právě svět, kde kdejaké „script kiddie“ představuje ohrožení systému, ujme nadstandardnímu bezpečnostnímu mechanismu onu nadstandardnost.

Zvýšení bezpečnosti Linuxu aplikací nadstandardních bezpečnostních mechanismů nepostihne pouze jej, nýbrž celý GNU Linux. Jde totiž i o zabezpečení uživatelských programů, které běží nad ním. Tyto **programy nejsou zpravidla tvořeny s cílem býti bezpečnými, nýbrž s cílem přinést maximální možný užitek**. Existence zranitelnosti v programu je důsledkem buďto nesprávné implementace správného algoritmu anebo prostě jen implementace nesprávného algoritmu. Program si tak za svou zranitelnost může sám a nejsnazší cestou k jejímu odstranění je jej nepoužívat a **napsat si** jeho **vlastní variantu zranitelností prostou**. Na to však **absolutně není čas**, chce-li provozovatel služby službu někdy také skutečně provozovat. Když si provozovatel spočítá, že se mu vývoj jemu potřebného vybavení prostého zranitelností nevyplatí, přistoupí ke všem ostatním na cestu nejmenšího odporu, kde se zranitelnosti tolerují a **hledá se nejméně odporu kladoucí řešení. A tím je provoz softwaru plného zranitelností zároveň s bezpečnostními technikami** zavádějícími taková opatření, která hrozby plynoucí z existence těchto zranitelností eliminují či minimalizují rozsah škod, který by mohly způsobit. To znamená, že v kódu dál tyto zranitelnosti zůstanou, nicméně nebude možné je zneužít. **Bezpečnostní techniky tak zpravidla působí čistě preventivně**.

Přijetí tolerance zranitelností na jednu stranu spoří člověkohodiny, které by bez ní byly potřeba na odstranění těchto zranitelností. Na druhou stranu však plýtvá instrukčními cykly, které se použijí na provoz preventivně působících bezpečnostních technik. **Tento stav je umožněn přebytkem výkonnosti dnešních systémů**. S neustálým růstem výkonnosti bude zřejmě i nadále vzrůstat význam preventivně působících bezpečnostních technik, neb díky výkonnostním přebytkům si systémy budou moci stále častěji a ve větší míře dovolit jejich provoz.

Zajímá-li čtenáře, jak velký dopad na výkonnost mají nadstandardní bezpečnostní mechanismy nyní, proti jakým hrozbám jeho systém ochrání či prostě jen hledá odpověď na otázku, zda-li se mu jejich nasazení vyplatí, potom je na dobré cestě. Dočítá totiž právě úvod dokumentu Zvýšení bezpečnosti Linuxu, který si vzal za svůj cíl analyzovat několik vybraných nadstandardních bezpečnostních řešení a na základě měření provedeném na systému, který pro účely měření poskytl CIV ZČU, odpovědět mimo jiné na výše uvedené otázky.

[1] Čili např. firewall.

[3] Například SMAP/SMEP (2012), Intel MPX (2013), RAP (2016) [Spe16], a řada dalších ...

2. Klasifikace bezpečnosti informačních systémů

Existence trhu s informačními systémy (IS) a poptávky po těchto systémech ze strany laiků (v oblasti bezpečnosti) vedou k poptávce po standardizaci měřítka uplatnitelného pro jejich porovnání. Zájemce o IS zpravidla nemá prostředky ani znalosti potřebné pro provedení (bezpečnostního) auditu všech IS na trhu. Ověření (bezpečnosti) IS proto uživatel často deleguje na na tvůrcích porovnávaných IS nezávislé autority, které IS zhodnotí a zároveň zaručí, že hodnocení bylo provedeno řádně. Hodnocení IS vyžaduje existenci objektivních a přesně definovaných kritérií bezpečnosti.

2.1 Kritéria pro hodnocení bezpečnosti IS

Kritéria pro zhodnocení bezpečnosti IS by měla poskytnout

- uživateli IS měřítko pro vyjádření stupně zabezpečení IS,
- výrobcům IS zpětnou odezvu a vodítko pro zlepšování jejich produktů a
- hodnotícím autoritám jasná hodnotící pravidla.

Kritérií pro hodnocení bezpečnosti IS je více. V rámci snah o zvýšení bezpečnosti Linuxu se má smysl bavit pouze o těch, která se váží na prostředí operačních systémů.

2.2 TCSEC

S vývojem kritérií pro hodnocení bezpečnosti IS se celosvětově začalo koncem šedesátých let. V USA byla za tímto účelem poprvé vytvořena pracovní skupina na tamějším ministerstvu obrany [Han93]. Mimo jiné dokumenty vzešly z ministerstva obrany USA poměrně široce známé standardy týkající se bezpečnosti, jejichž přehled lze nalézt na `http://en.wikipedia.org/wiki/Rainbow_Series`.

- TCSEC – „Orange Book“
- TDI – „Grey Book“
- TNI – „Raspberry Book“

Hodnocení bezpečnosti IS se věnuje dokument TCSEC, který byl zveřejněn v polovině osmdesátých let. Jedná se o jeden z prvních veřejně publikovaných dokumentů definujících kritéria pro hodnocení bezpečnosti IS. Tato kritéria jsou formulována velmi obecně a lze podle nich hodnotit jakoukoliv komponentu IS.

2.2.1 Kategorie IS dle TCSEC

TCSEC definuje pro klasifikaci IS (a tedy i systému operačního (OS)) divize

- A – minimální ochrana,
- B – volitelná ochrana,
- C – povinná ochrana a
- D – verifikovaná ochrana

dále se dělící na třídy [85]. IS lze zařadit do jedné z následujících kategorií respektive tříd vyjadřujících úroveň bezpečnosti IS.

2.2.1.1 Třída D – minimální ochrana

Třída D zahrnuje IS s žádnými či pouze minimálními prvky bezpečnosti. Minimálními prvky bezpečnosti se v tomto případě rozumí prvky, které IS nestačí pro zařazení do třídy C1. Do třídy D se řadí například MS-DOS a operační systémy jej obsahující – Windows až do Millenium Edition včetně.

2.2.1.2 Třída C1 – volitelná ochrana

Zařazení IS do třídy C1 předpokládá

oddělení uživatelů a dat,
: Přístup subjektů k objektům je možné řídit respektive není neomezený.

autentizaci subjektů,
: Každý subjekt pracující se systémem musí být autentizován (zpravidla heslem). Třída C1 si nicméně neklade požadavek na to, aby každý reálný uživatel (ve smyslu osoba) byl v systému veden jako individuální subjekt. IS musí chránit autentizační data před neautorizovaným přístupem.

ochranu domény před vnějšími zásahy a
: IS musí zajistit ochranu své domény před vnějším zásahem, kterým může být např. modifikace kódu či dat.

volitelné řízení přístupu (DAC).
: Řízení přístupu založené na existenci ACL pro každý objekt v systému. ACL je definováno vždy subjektem vlastnícím daný objekt. **Řízení přístupu k objektu je tak zcela v režii jeho vlastníka.**

Do této kategorie lze zařadit GNU Linux systémy [Pat].

2.2.1.3 Třída C2 – řízená ochrana přístupu

Třída C2 požaduje po IS

vše, co požadovala třída C1,

jemněji konfigurovatelné nepovinné řízení přístupu (DAC),
: Oprávnění musí být možné definovat na úrovni každého jednotlivého uživatele.

rušení obsahu systémových prostředků při jejich znovupoužití,
: IS musí zajistit, aby při opakovaném použití libovolného prostředku v rámci systému nebylo možné z něj získat jeho původní obsah. Obsah nesený libovolnými systémovými prostředky (zejména hlavní pamětí) je před přístupem subjektu vždy zrušen/inicializován. ⇒ Subjekt nemá možnost získat přístup k datům subjektu využívajícího tentýž prostředek před ním.

jednoznačnou identifikaci a autorizaci každého jednotlivého uživatele a
: Každý jednotlivý uživatel IS musí mít v rámci IS jednoznačnou identitu. S touto identitou systém pojí všechny kontrolovatelné kroky, jež může jednotlivec v rámci IS vykonat. K autentizaci jsou v tomto případě nutné minimálně 2 informace – uživatelské jméno a heslo.

logování z hlediska bezpečnosti významných událostí.
: Veškeré akce týkající se přístupu k objektům musí být IS schopen zaznamenávat. Záznamy jsou chráněny před neoprávněným přístupem. Mezi zaznamenávané události patří

  - úspěšné i neúspěšné autentizace subjektů,
  - zavedení/odstranění objektů do/z adresového prostoru subjektu a
  - akce provedené správci IS.

  Pro každou zaznamenávanou událost je nutné vést

  - datum a čas události,
  - typ události a
  - úspěch/neúspěch události.

  IS musí správci systému poskytnout možnost provádět audit selektivně pro jednotlivé typy událostí a též pro jednotlivé subjekty. Účelem tohoto opatření je informovat o pokusech subjektů o neautorizované akce.

2.2.1.4 Třída B1 – značkovaná ochrana

Třída B1 přepokládá

splnění požadavků třídy C2 ,

klasifikaci subjektů a objektů podle stupně jejich ochrany,
: Subjekty jsou asociovány se značkami reprezentujícími stupeň jejich ochrany. Tytéž značky jsou přiděleny objektům danými subjekty vlastněnými. Značky jsou hierarchicky uspořádány.

povinné řízení přístupu (MAC) a
: Na základě značek jsou prováděna rozhodnutí o přidělení/nepřidělení přístupových oprávnění. Jedná se o vynucené řízení přístupu, které má v moci distributor těchto značek. Subjekt tak v rámci MAC nerozhoduje o přístupových právech k jemu vlastněným objektům. Subjekt má přístup pouze k objektům s nižší klasifikační značkou.

existenci neformální definice bezpečnostní politiky.
: Musí existovat neformální popis implementovaného bezpečnostního modelu.

2.2.1.5 Třída B2 – strukturovaná ochrana

Třída B2 předpokládá

splnění požadavků třídy B1 ,

formální definici modelu bezpečnostní politiky IS,
IS je podle tohoto modelu implementován.
MAC nad všemi subjekty a objekty v systému a
Povinné řízení přístupu musí být povinně uplatňováno nad všemi v systému přítomnými subjekty a objekty.
dekompozici IS do modulů.
IS musí být vnitřně strukturován do do značné míry nezávislých modulů. Rozlišují se moduly citlivé na bezpečnost a moduly ostatní. Jednotlivé moduly musí mít nejnižší možné oprávnění, které jim zaručí správný chod. K oddělení modulů citlivých na bezpečnost musí IS efektivně využívat možností jemu dostupného hardwaru. Příkladem je mechanismus segmentace oddělující jednotlivé objekty v adresním prostoru a vynucující přístupová oprávnění nad nimi.

2.2.1.6 Třída B3 – bezpečnostní domény

IS zařazený do třídy B3

splňuje požadavky kladené na IS třídou B2 ,

přenechává jakékoliv autorizace referečním monitorům,
Referenční monitor (RM) je komponenta systému zprostředkovávající autorizaci všech přístupů subjektů k objektům. RM musí [Pat] být
- odolný vnějším zásahům (útokům),
- přítomný **u každého rozhodování** o přístupu a
- dostatečně malý na to, aby mohl být podroben analýze a testování.

RM tak vždy obsahuje pouze nezbytně nutný kód pro realizaci stanovené bezpečnostní politiky.
je schopný se zotavit po útoku a

Zotavení si nesmí vyžádat žádné ústupky z hlediska bezpečnostní politiky.
definuje roli bezpečnostního správce.
Oblasti nastavení týkající se bezpečnosti vyžadují roli administrátora zabezpečení. Řadoví administrátoři mohou konfigurovat zabezpečení pouze po nabytí role bezpečnostního správce, které je logováno.

2.2.1.7 Třída A1 – ověřená ochrana

Třída A1 představuje v klasifikační hierarchii TCSEC nejvyšší úroveň bezpečnosti. Zařazení IS do třídy A1 předpokládá

splnění požadavků třídy B3 a

formální důkaz všech funkčních požadavků nižších tříd.
IS disponuje analýzou implementace funkčních požadavků nižších tříd a formálním důkazem jejich funkčnosti. To zajistí vysoký stupeň jistoty toho, že IS je z bezpečnostního hlediska realizován správně.

3. Základní pojmy

Pro bližší seznámení s bezpečnostními aspekty Linuxu je vhodné se nejprve seznámit se základními pojmy, hrajícími v problematice bezpečnosti hlavní role.

3.1 Uživatel v Linuxu

Uživatel je subjekt operující v rámci systému pod unikátním identifikátorem User Identifier (UID). Každý uživatel v rámci systému

- musí mít jméno (username),
- může mít definována jedinečná přístupová oprávnění,
- může být vlastníkem souborů,
- musí mít stanoven identifikátor primární skupiny *Primary GID (PGID)* čili primární skupinu, které je členem a
- může mít definován seznam sekundárních skupin, jejichž je členem.

Různí uživatelé by měli mít navzájem různá UID, nicméně nutné to není. V případě stejných UID budou jejich přístupová oprávnění totožná.

3.1.1 Superuživatel

Superuživatel je uživatel s UID rovným 0. Takový uživatel má v rámci systému neomezená oprávnění k jakékoliv činnosti. V rámci systému jich může být i více a označují se jako privilegovaní. Ostatní uživatelé jsou neprivilegovaní.

3.2 Uživatelský účet

Uživatelský účet je množina nesourodých položek vázaných na právě jednoho uživatele. Zahrnuje vše k uživateli vztažené:

- identifikátor UID,
- identifikátor primární skupiny PGID,
- seznam sekundárních skupin,
- domovský adresář a
- přihlašovací shell.

V systému existují dva druhy uživatelských účtů – *normální* a *systémový*. Toto dělení není závazné, nýbrž jedná se pouze o zvyk určující mapování UID normálních účtů do intervalu $\langle 1000, 2^{16} \rangle$ a účtů systémových do $\langle 0, 1000 \rangle$. Do systémových uživatelských účtů se ve většině případů nelze přihlásit, neboť jejich přihlašovacím programem nebývá shell.

3.3 Skupina

Uživatelé jsou organizováni v množinách zvaných *skupiny*. *Skupina* je množina uživatelů v rámci systému identifikovaná prostřednictvím Group Identifier (GID). Umožňuje specifikovat přístupová oprávnění uživatelům v ní obsaženým. Každý uživatel musí být členem nejméně jedné skupiny. Jedna ze skupin uživatele je vždy primární. Uživatel má možnost svou primární skupinu změnit příkazem `newgrp` na jinou skupinu z množiny skupin, jichž je členem.

3.4 Soubor

Soubor je reprezentací konkrétního prostředku (obyčejný soubor, adresář, speciální soubor, síťový socket, ...) v rámci počítače, s kterým je možné manipulovat – číst z něho či do něj zapisovat. V prostředí operačního systému GNU/-Linux je většina prostředků reprezentována *soubory*, z nichž sestává stromová struktura zvaná souborový systém. V souborovém systému je soubor jednoznačně identifikován nejčastěji číslem svého inode. V následujícím textu se „soubor“ rovná jakémukoliv prostředku operačního systému – tedy nikoliv výhradně obyčejný soubor.

3.5 Program

Program je soubor obsahující posloupnost strojových instrukcí zakódovaných do binární podoby srozumitelné procesoru.

3.6 Proces

Proces je n-tice sestavená z nestejnorodých prvků, z nichž nejvýznamnější jsou

program,
: Spustitelný soubor.

$k \times$ **program counter,**
: Počítadlo instrukcí pro každé z k vláken operujícím nad adresovým prostorem programu.

VAS,
: Adresní prostor paměti, který je uspořádán jinak nebo je dokonce větší, než je fyzicky připojená hlavní paměť. Podrobnější zmínka o VAS je uvedena v kapitole 4.1.2.

Process Identifier (PID) a
: Jednoznačný identifikátor procesu v rámci počítače.

RI a EI.
: Reálná a efektivní identity procesu, od nichž se odvíjí oprávnění procesu. Podrobnější zmínka o identitách procesu je uvedena v kapitole 3.6.3

3.6.1 Vlastník procesu

Vlastník procesu je uživatel, který inicioval spuštění procesu. Tento uživatel je oprávněn řídit jeho běh prostřednictvím signálů. Zasílat signály procesu mohou

- proces s RUID rovným RUID či SSUID adresáta,
- proces s EUID rovným RUID či SSUID adresáta a
- proces s capabilitou `CAP_KILL` – např. superuživatelský proces.

Ostatní nemohou s procesem prostřednictvím signálů komunikovat.

3.6.2 Skupinový vlastník procesu

Skupinový vlastník procesu je skupina, kterou měl vlastník procesu nastavenu jako primární v okamžiku vytvoření procesu.

3.6.3 Identita procesu

Identita procesu je uspořádaná dvojice identifikátorů – (UID,GID). Rozhodně proces jednoznačně neidentifikují (od toho je PID). Veškeré atributy ovlivňující identity konkrétního procesu je možné zobrazit příkazem `ps` (viz příkaz 3.1). Každý proces má dvě identity – *reálnou* (()) a *efektivní* (()).

```
$ ps -o ruid,rgid,uid,gid,comm --pid
RUID  RGID   UID   GID COMMAND
1000    24  1000    24 bash
```

Příkaz/y 3.1: Získání atributů určujících identitu aktuálního příkazového interpretru

3.6.3.1 Reálná identita procesu

Reálná identita (RI) procesu je uspořádanou dvojicí identifikátorů

- Real UID (RUID) – UID vlastníka procesu
- a Real GID (RGID) – GID skupinového vlastníka procesu.

Reálná identita ovlivňuje oprávnění procesu zasílat signály jiným procesům. Proces s reálnou identitou neprivilegovaného uživatele může zasílat signály pouze procesům se stejnou reálnou identitou. Procesy standardně dědí oprávnění svého předka, díky čemuž rodič a jeho potomci mají možnost komunikovat.

3.6.3.2 Efektivní identita procesu

Efektivní identita (EI) procesu je uspořádanou dvojicí identifikátorů Effective UID (EUID) a Effective GID (EGID). Standardně je efektivní identita tatáž, jako reálná identita. Efektivní identita určuje

- přístupová oprávnění procesu k existujícím souborům
- a vlastníka (též skupinového) procesem nově vytvářených souborů.

3.6.3.3 Změna identity

Po volání `fork()` má nový proces stejné RI i EI, jako jeho rodič.

Pokud by měly všechny procesy v systému tytéž identity, bezpečnost by byla podmíněna bezpodmínečně dobrými úmysly a kompetencí každého procesu nakládat se soubory, s kterými nakládá. Možnost změny identity je tedy nanejvýš nutná. Proces může změnit svou identitu voláním

`exec(/bin/login,…)`
: Po případném úspěšném přihlášení uživatele se RI i EI nastaví na hodnoty UID respektive PGID uživatele.

`exec(<SPUSTITELNY_SOUBOR_S_NASTAVENYM_SUID_BITEM>,…)`
: Po ověření oprávnění procesu daný soubor spustit se změní EI (konkrétně EUID) procesu na UID vlastníka spouštěného souboru.

`exec(<SPUSTITELNY_SOUBOR_S_NASTAVENYM_SGID_BITEM>,…)`
: Po ověření oprávnění procesu daný soubor spustit se změní EI (konkrétně EGID) procesu na GID skupinového vlastníka spouštěného souboru.

`setuid(<POZADOVANE_UID>)` a má-li volající EUID rovno 0
: Vždy dojde ke změně EUID i RUID procesu na požadovanou hodnotu.

`setuid(<POZADOVANE_UID>)` a má-li volající EUID různé od 0
: Po ověření skutečnosti, že `<POŽADOVANÉ_UID>` je rovno RUID nebo SSUID, se provede změna EUID procesu na požadovanou hodnotu.

`setgid()`, `seteuid()`, `setegid()`, …
: Obdobný postup jako při volání `setuid()`.

3.6.3.4 SUID a SGID bity

Je zjevné, že nezanedbatelnou roli v procesu změny efektivní identity (EI) mohou hrát tzv. Set UID (SUID) a Set GID (SGID) bity. Tyto bity změní EI procesu, který vykonává program (spustitelný soubor). Pokud je spustitelným souborem skript, příznaky SUID a SGID nemají vliv, neboť jejich spouštění je pouze chimérou pro zjednodušení jejich použití uživateli. Rozhodující je v tomto případě (ne)nastavení těchto příznaků souboru představujícího jejich interpretr.

3.7 Režimy procesoru

Režimy procesoru slouží k rozlišení úrovně oprávnění procesorem vykonávaného kódu. Jejich počet je dán architekturou procesoru. Při zanedbání existence paravirtualizace lze říci, že pro chod operačního systému postaveného na Linuxu stačí mít k dispozici právě dva režimy.

3.7.1 Režim jádra

Režim jádra – *kernel mode (KM)* – někdy též označovaný jako „privilegovaný režim“ je stav procesoru, ve kterém procesor vykonává strojový kód jádra. Může tak činit

v *kontextu procesu (KM-PC)* **nebo**
: Vykonávání strojového kódu jádra mající vazbu na konkrétní proces. Nejčastější příčinou přepnutí do KM-PC je vykonání systémového volání obsaženého ve strojovém kódu uživatelského procesu.

mimo *kontext procesu (KM-NPC)*.
: Vykonávání strojového kódu jádra nemající vazbu na konkrétní proces. Typicky se může jednat například o obsluhu přerušení.

Každopádně v obou případech KM lze využívat všech architekturou poskytovaných instrukcí. Jakákoliv operace vyžadující přístup do VAS uživatelského procesu musí být provedena v jeho kontextu – v KM-PC.

3.7.2 Uživatelský režim

Uživatelský režim (UM) někdy též označovaný jako „neprivilegovaný režim“ je režim, kdy procesor vykonává strojový kód uživatelského procesu. V tomto režimu

- jsou zapovězeny některé instrukce a
- přístupná je pouze část hlavní paměti označovaná jako Low Memory (LM).

3.8 Hlavní paměť

Hlavní paměť počítače je souvislé konečné pole slabik, kde každá jedna slabika má unikátní celočíselnou adresu – index v poli, na kterém se nalézá.

4. Správa paměti

Následující text uvažuje výhradně počítače založené na *Von Neumannově architektuře*, tedy počítače, které

- **nerozlišují mezi strojovými instrukcemi a daty (alespoň co se týče jejich uchovávání)**,

  Obé představuje ve Von Neumannově architektuře sekvenci binárních hodnot, která je ukládána do hlavní paměti.
- **paměť je rozdělena na úseky stejné velikosti – buňky** a
- **pořadové číslo buňky představuje její adresu**.

Seznámení s mechanismy správy hlavní paměti je nezbytné k objasnění principů, na kterých stojí podstatná část vylepšení jádra dodaných mu patchem[1] Grsec/PaX.

Správa paměti, to je především překlad adres. Překladem adres je v tomto případě zobrazení

$$\text{logická adresa} \longrightarrow \text{fyzická adresa}$$

Vzhledem k tomu, že k překladu dochází při každém přístupu do paměti a že se jedná o relativně složitou operaci, je správa paměti implementována částečně hardwarově (viz kapitola 4.2.2). GNU Linux je víceúlohový systém podporující multitasking disponující hlavní pamětí adresovanou fyzickými adresami. Problém, jak poskytnout každému z obecně n procesů v systému běžících adresní prostor začínající nulou řeší segmentace (viz kapitola 4.3), případně stránkování (viz kapitola 4.4.1). Problém, jak poskytnout procesům více paměti, nežli je v systému dostupné paměti řeší stránkování.

4.1 Adresní prostory

Adresní prostor (AS) je označení pro souvislý rozsah adres. V oboru správy paměti lze hovořit o

- **Physical Address Space (PAS)**,

  Adresní prostor, jehož délka odpovídá kapacitě v počítači instalované hlavní paměti [Dud04]. Tvoří jej *fyzické adresy (PA)*. V dnešních systémech je typicky řádově menší nežli logický/virtuální adresní prostor. Adresní prostor, jehož délka odpovídá kapacitě v počítači instalované hlavní paměti [Dud04]. Tvoří jej *fyzické adresy (PA)*. V dnešních systémech je typicky řádově menší nežli logický/virtuální adresní prostor.
- **Virtual Address Space (VAS)**,

  Adresní prostor paměti, který je uspořádán jinak nebo je dokonce větší, než je fyzicky připojená hlavní paměť.
- **Logical Address Space (LAS)** a

  Adresní prostor tvořený logickými adresami.
- **Address Space (AS)**.

  Adresní prostor vyhrazený pro jeden proces. Obecně je jím myšlen zpravidla VAS.

4.1.1 Logický adresní prostor

Logický adresní prostor (LAS) je adresní prostor používaný procesorem čili adresy jej pokrývající – logické adresy (LA) – jsou součástí strojového kódu. Institut logického adresního prostoru existuje kvůli podpoře segmentace[2].

4.1.1.1 Logická adresa

Logická adresa (LA) je uspořádaná dvojice

- **selektor segmentu** a

  Identifikátor segmentu, v němž se paměťové místo nachází.
- **offset v rámci daného segmentu**.

  Posunutí paměťového místa vzhledem k počátku segmentu, ve kterém leží.

Délka LA určuje maximální velikost paměti, kterou je procesor schopen adresovat. Je minimem z

[1] *Grsecurity/PaX* je soubor modifikací jádra posilujících bezpečnost distribuovaný právě formou patche jádra.
[2] Segmentace je pro architekturu x86 povinná.

- délky slova[3] použité procesorové architektury a
- šířky adresové sběrnice procesoru.

Je zřejmé, že při LA $= n$ je velikost LAS $= 2^n - 1$ slabik. Proces, jemuž LAS náleží, nemá možnost jej využívat celý. LAS je rozdělen na

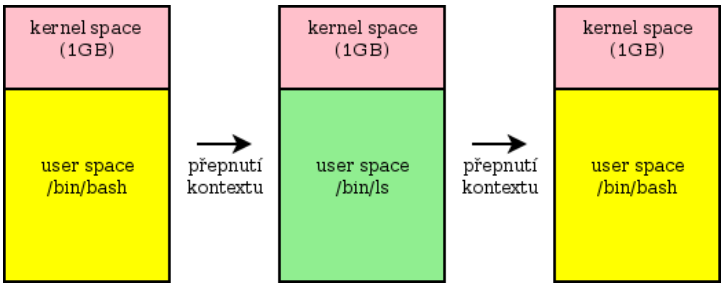
4.1.1.2 Kernel Space

Kernel Space (KS) neboli systémová část logického adresního prostoru je adresní rozsah v LAS nepřístupný strojovému kódu běžícímu v user mode (UM) čili nepřístupný uživatelským procesům. Naopak strojovému kódu běžícímu v KM – jádru – je tento prostor přístupný. Někdy je tento prostor označován též jako *High Memory (HM)*.

Zatímco každý proces přítomný v systému disponuje svým vlastním LAS, jádro žádný vlastní LAS nemá. Namísto toho využívá určitou část LAS každého z procesů nazývanou Kernel Space (KS). Kód jádra a veškerá jím používaná data jsou mapována na stejné místo v rámci jakéhokoliv LAS (viz obrázek 4.1), respektive všechny LAS mají část adresního prostoru – Kernel Space – mapovanou na tytéž stránky paměti. Z toho plyne, že adresní prostor jádra je v paměti umístěn stále na týchž adresách nezávisle na kontextu procesu. Jádro nemůže být z hlavní paměti nikdy odstránkováno[4] a KS lze díky jeho trvalé přítomnosti v každém LAS využívat pro ukládání informací meziprocesové komunikace. Do Kernel Space mají přístup

jak vlákna vykonávající kód jádra v kontextu procesu
: Vlákna vykonávající činnost vyvolanou určitým procesem.

tak i vlákna vykonávající kód jádra mimo kontext procesu.
: Vlákna vykonávající činnost, která se nevztahuje k žádnému určitému procesu. Například vlákno plánovače procesů.



Obrázek 4.1: Přepnutí kontextu

4.1.1.3 User Space

User Space (US) neboli uživatelská část logického adresního prostoru je adresní rozsah v LAS přístupný strojovému kódu běžícímu v user mode (UM) – uživatelskému procesu – stejně tak, jako je přístupný strojovému kódu běžícímu v KM-PC. Někdy je tento prostor označován též jako Low Memory (LM).

Oddělení od Kernel Space části LAS je realizováno mechanismem segmentace (viz kapitola 4.3).

4.1.2 Lineární/virtuální adresní prostor

Lineární adresní prostor či též *virtuální adresní prostor (VAS)* je adresní prostor umožňující realizaci mechanismu virtuální paměti (viz kapitola 4.4). Tvoří mezistupeň mezi LAS a PAS. Vyskytuje se pouze v případě nasazení mechanismu virtualizace paměti v systému – mechanismu stránkování paměti (viz kapitola 4.4.1).

4.2 Implementace správy paměti

Správa paměti je natolik komplexní, že její implementace je částečně hardwarová (viz kapitola 4.2.2) a částečně i softwarová (viz kapitola 4.2.1).

4.2.1 Memory Manager

Memory manager je komponentou jádra mající na starosti správu paměti. Jeho pracovní náplní jsou přidělování a odebírání paměti procesům, sdílení paměti mezi procesy, přesun stránek mezi hlavní pamětí a pamětí externí v případě nedostatečné kapacity hlavní paměti, převod mezi logickou (LA) a fyzickou (PA) adresou a mnohé další činnosti týkající se hlavní paměti. Hlavním úkolem memory manageru je zajištění chodu jak jádra tak i procesů běžících v systému s menším fyzickým adresním prostorem, nežli s jakým systém disponuje.

[3] Délka slova je například pro architekturu x86-32 rovna 32 bitům.

[4] Přesunuto mechanismem stránkování z hlavní do externí paměti.

4.2.2 Memory Management Unit

Memory Management Unit (MMU) je hardwarová komponenta umístěná v rámci procesoru[93] zodpovědná za zobrazení

$$\text{logická adresa} \longrightarrow \text{fyzická adresa}$$

, přičemž při nasazení stránkování se jedná dokonce o složené zobrazení

$$\text{logická adresa} \longrightarrow \text{lineární adresa} \longrightarrow \text{fyzická adresa}$$

V každém případě účelem MMU je nalezení fyzické adresy. Pro provedení zobrazení využívá MMU informací získaných z tabulky stránek nebo z jejích cache v podobě Translation Lookaside Buffer (TLB).

4.3 Segmentace

Segmentace je mechanismus správy AS procesu, který jej dekomponuje na více adresních prostorů tzv. segmentů.

Segment je samostatný adresní prostor mapovaný do AS procesu.

Segmentová tabulka je datová struktura uložená v hlavní paměti počítače. Její úlohou v procesu segmentace je provádět zobrazení

- logická adresa $\rightarrow$ fyzická adresa není-li na systému nasazena technika stránkování či
- logická adresa $\rightarrow$ lineární adresa je-li na systému nasazena technika stránkování.

Obsah deskriptoru segmentu, kterýžto je uložen v tabulce deskriptorů segmentů, se odvíjí od toho, zda-li je v systému aktivní mechanismus stránkování nebo není.

Každý záznam vedený v segmentové tabulce nese

číslo segmentu,
: Identifikátor segmentu.

bázovou adresu segmentu,
: PA určující umístění segmentu v rámci PAS.

délku segmentu a
: Délka segmentu se může během vykonávání měnit. Délky různých segmentů jsou obecně různé, neboť jsou odvislé od velikosti obsahu jimi neseného.

přístupová práva k segmentu.
: Přístup k jednotlivým segmentům je možné řídit nezávisle na ostatních segmentech. Informace o přístupových oprávněních jsou uloženy v segmentové tabulce v podobě příznaků

  read-only a
  : Příznak zabraňující zápisu do segmentu.

  execute.
  : Příznak zabraňující spuštění obsahu segmentu.

4.4 Virtualizace paměti

Každý proces v systému disponuje svým LAS, zatímco PAS je v systému pouze jediný a konečný. Má-li systém umožnit běh „nekonečně mnoho“[5] procesů (což je obecně požadováno), musí v něm existovat mechanismus, který omezení dané nedostatečnou kapacitou PAS obejde. Říká se mu *virtualizace paměti*.

Virtualizace paměti je technika, která umožňuje přidělit procesům výrazně více paměti, než kolik jí má systém reálně k dispozici. Virtualizaci zajišťuje tzv. stránkování (viz kapitola 4.4.1). K virtualizaci paměti se vážou i další benefity a to

ochrana paměti,
: Paměťové prostory v systému běžících procesů jsou od sebe dokonale izolovány.

zjednodušení paměťového modelu,
: Eliminuje potřebu relokace.

snížení nebezpečí uváznutí a
: Přidává možnost odebrat procesu již přidělenou část PAS.

[5] „Nekonečně mnoho“ procesů je v tomto případě nadnesené vyjádření pro 2^{15} procesů, což je limit, dnes běžně nastavený, v operačních systémech stavěných nad Linuxem. Uveden je v souboru `/proc/sys/kernel/pid_max`. Lze jej zvýšit zápisem do daného souboru až na 2^{22} procesů. V porovnání s omezením daným velikostí PAS se nicméně i 2^{15} zdá být „v nekonečnu“.

úspora zdrojů (hlavní paměti).
Do paměti se nahrávají pouze procesem skutečně používané stránky a tím se výrazně šetří nepřidělenými rámci. Do PAS se díky tomu vejde najednou nepoměrně více procesů.

4.4.1 Stránkování

Stránkování je mechanismus realizující virtualizaci paměti. Zavádí VAS jako prostředníka mezi LAS a PAS. Na architektuře x86 je stránkování nepovinné, nicméně Linux jej ke svému běhu vyžaduje, neboť přidělení více paměti, nežli je v systému reálně dostupné paměti, je dnes naprostým standardem. Stránkování předpokládá následující skutečnosti:

- VAS každého procesu je rozdělen na úseky pevné délky[6] zvané *rámce*.
- PAS je rozdělen na úseky pevné délky zvané *stránky*.
- Velikost rámce je rovna velikosti stránky[7].
- Existuje vyhrazený prostor ve vnější paměti, kde mohou být uloženy stránky – tzv. odkládací prostor neboli SWAP.

Při každém přístupu procesoru do hlavní paměti je nutné provést překlad VA → PA. Překlad adres probíhá zcela automaticky díky komponentě procesoru zvané Memory Management Unit (MMU). Ta při překladu používá tabulku stránek respektive její vyrovnávací paměť – Translation Lookaside Buffer. Stránky je možné mapovat na rámce naprosto nahodile, proto **mapování VA na PA nemusí být nutně lineární**. VA je v případě nasazení stránkování vnímána jako uspořádaná dvojice

číslo stránky a
Index do tabulky stránek.
offset.
Odstup přistupované adresy od počátku stránky respektive rámce. Tato komponenta VA se při překladu adres pouze zkopíruje do PA.

Z pohledu procesu je mechanismus fragmentace VAS na stránky zcela transparentní. Proces vnímá VAS jako jediný souvislý adresní prostor.

4.4.1.1 Výpadek stránky

Výpadek stránky nebo též *Page fault* je výjimka předávaná komponentou MMU jádru, čili Linuxu k obsluze. Při nasazení PaXu v systému může mít tato výjimka obecně různé významy, které shrnuje tabulka 4.1. Obsluhu zajišťuje funkce jádra `do_page_fault()`. **Modifikací této funkce lze do systému dodatečně doplnit řízení přístupu do paměti**, jak činí například Page eXec project. Při každém výskytu výjimky Page fault pak tato funkce vyhodnocuje zda-li se opravdu jedná o výpadek stránky v původním smyslu či byl v daném případě pouze přetížen pro jiný smysl (viz tabulka 4.1).

Použití	Popis významu
výpadek stránky (původní význam)	Stránka nebyla v paměti nalezena či oprávnění není dostatečné. Je potřeba ji nahrát do paměti. Dojde k přerušení vykonávání na procesoru vyvolané komponentou MMU v situaci, kdy v hlavní paměti nebyl nalezen rámec odpovídající hledané stránce.
PAGEEXEC (přetížený supervisor bit)	Technika PAGEEXEC přetěžuje význam supervisor bitu v tabulce stránek – dává mu význam ne/spustitelnosti stránky. Při jakémkoliv přístupu ke stránce kódem běžícím s oprávněním Ring 0 je vyvolána výjimka Page fault, přestože rámec stránce odpovídající je v paměti nahrán.
SEGMEXEC (přetížený mechanismus segmentace)	Technika SEGMEXEC přetěžuje mechanismus segmentace, jehož původní, Linuxem nepoužívaný význam rozdělení AS procesu na segmenty znovu zavádí do praxe, čímž dosáhne odlišení datového a instrukčního přístupu do paměti (při překladu LA → VA).

Tabulka 4.1: Různé významy výjimky Page fault

[6] Délka rámce/stránky je závislá na použité procesorové architektuře.
[7] Velikost rámce respektive stránky je při použití procesoru architektury x86-32 typicky 4 kiB, ale může být i více

4.4.1.2 Tabulka stránek

Tabulka stránek je datová struktura umístěná v hlavní paměti. Každý proces disponuje svou vlastní tabulkou stránek. Tabulka stránek obsahuje záznamy o přidělených rámcích hlavní paměti. Každý jeden záznam v PT obsahuje mimo jiné

- číslo rámce,
- číslo stránky mapované do daného rámce (nebo informaci, že je rámec volný),
- příznak nesoucí informaci o tom, zda-li je obsah umístěný na stránce zapisovatelný,
- příznak nesoucí informaci o tom, zda je stránka přítomna v PAS či
- příznak nesoucí informaci o tom, zda stránka byla modifikována.

Fyzická adresa tabulky stránek je uložena ve speciálním registru PTBR procesoru a přístup k ní se tak obejde bez překladu adres. Překlad VA $\rightarrow$ PA prováděný v tabulce stránek spočívá v nahrazení čísla stránky číslem rámce. Offset zůstává vzhledem k též délce rámců a stránek zachován.

4.4.1.3 Transaction Lookaside Buffer

Translation Lookaside Buffer (TLB) je cache, do níž se ukládají naposledy přistupované záznamy z tabulky stránek. MMU z ní získává informace potřebné ke správě VAS. Pokud potřebné informace nenalezne, jsou do TLB nahrány z tabulky stránek. Na architektuře x86 jsou od procesorů Intel Pentium a AMD Duron k dispozici hned dvě TLB, které pracují nezávisle na sobě:

- *Instrukční TLB (ITLB)* je vyrovnávací paměť nesoucí informace o nejčastěji přistupovaných stránkách nesoucích programem vykonávaný strojový kód.
- *Datová TLB (ITLB)* je vyrovnávací paměť nesoucí informace o nejčastěji přistupovaných stránkách nesoucích programem zpracovávaná data.

4.4.1.4 Velikost tabulky stránek

Tabulka stránek každého běžícího procesu musí být nahrána v hlavní paměti celá po celou dobu jeho existence. Nechť

n	je počet uživatelských procesů běžících v systému,
w [b]	je šířka VA,
p [B]	je délka jedné stránky,
r [B]	je délka jednoho záznamu v tabulce stránek a
x [B]	je celková velikost tabulky stránek.

Celková velikost všech tabulek stránek přítomných v systému – x – je rovna výrazu 4.1. Výpočet x za předpokladu,

$$\frac{2^w}{p} \cdot n \cdot r \quad \text{[B]}$$

Výraz 4.1: Celková velikost všech tabulek stránek v systému

že

- $n = 32$ v systému běží 32 procesů,
- $w = 32$ [b] délka VA je 32 bitů,
- $p = 2^{12}$ [B] délka jedné stránky je 4 KiB $= 2^{12}$ B a
- $r = 4$ že délka jednoho záznamu v PT je 4 B

je uveden v rovnosti 4.1. Tabulku stránek žádného procesu nelze odložit do externí paměti. Výsledek rovnosti 4.1 říká,

$$\begin{aligned} x &= \frac{2^{32}}{2^{12}} \cdot 2^5 \cdot 2^2 \quad \text{[B]} \\ x &= 2^{27} \quad \text{[B]} \\ x &= 131072 \quad \text{[KiB]} \end{aligned}$$

Rovnost 4.1: Výpočet celkové velikosti všech tabulek stránek v systému

že pro 32 procesů je zapotřebí prostor cca 130 MB. Pro snížení objemu tabulek stránek je možno použít víceúrovňové stránkování, inverzní tablku stránek či segmentaci.

4.4.1.5 Sdílení stránek

Procesy běžící v systému mohou používat tytéž

staticky linkované knihovny a
: V případě staticky linkovaných knihoven nedochází k úspoře místa v hlavní paměti, neboť kód těchto knihoven je zapsán při kompilaci programu mezi jeho strojové instrukce a to – pochopitelně – u každého programu na obecně různá místa.

dynamicky linkované knihovny.
: Strojový kód dynamicky linkovaných knihoven je vkládán do VAS procesu až v čase běhu a to do oblasti MMS (MMS) umístěné na samostatných stránkách VAS. Zde je potenciální prostor pro úsporu prostoru v hlavní paměti, neboť je zbytečné mít v paměti stejné kopie knihoven zvlášť pro každý proces.

Sdílení vybraných stránek VAS je realizováno tak, že tabulky stránek různých procesů pro určité své stránky odkazují na tentýž rámec PAS. Mezi procesy, jež jsou instancemi téhož programu lze tímto způsobem sdílet více než jen dynamicky linkované knihovny – též data i strojový kód. Každá další kopie stejného procesu pak zabere v paměti jen zanedbatelné množství paměti. Obecně platí, že sdílené stránky by měly umožňovat čtení (případně vykonávání) nikoliv však zápis jejich obsahu. Sdílení rámců je třeba zohlednit při odkládání paměti do externí paměti, neboť odložení sdílených dat ovlivní více než jeden proces.

4.5 Správa paměti v praxi

Linux segmentaci nevyužívá. Aby však mohl běžet na x86 procesorech, je nucen segmentaci alespoň formálně používat, neboť segmentace je na procesorech architektury x86 implementována povinně. Běžně se používají

- kódový segment pro kernel mode,
- datový segment pro kernel mode,
- kódový segment pro user mode a
- datový segment pro user mode.

Obsah kódových segmentů je povoleno spouštět a obsah datových segmentů je povoleno zapisovat. Ze všech segmentů je povoleno číst. Vše ostatní je zakázáno. Všechny segmenty jsou mapovány do téhož prostoru (začínají na adrese 0) a tudíž se překrývají. Selektor segmentu má tak vliv pouze na stanovení úrovně oprávnění. Adresa je plně určena částí logické adresy zvané „offset v rámci segmentu“.

Linux naopak využívá mechanismus stránkování a proto nejstarším jím podporovaným procesorem je i386, neboť ten přišel s podporou stránkování jako vůbec první x86-kompatibilní procesor.

4.6 Rozdělení AS procesu z hlediska obsažených dat

Do adresního prostoru procesu se mapují v kapitolách 4.6.1 až 4.6.7 popisovaná mapování. Všechna v nich uvedená mapování jsou sdílená mezi více vlákny. Mapování se obecně rozlišují dvojího typu:

- *Anonymní mapování* je mapování, pro které neexistuje předloha v rámci FS, nýbrž dané vzniká vždy prázdné při volání funkce `execve()` (při vytváření procesu).
- *Neanonymní mapování* je přesným opakem mapování anonymního. Existuje pro něj předloha ležící na filesystému.

Mapování lze též kategorizovat dle času jejich vzniku:

- *Mapování vzniklá při vytváření procesu* jsou mapování, která vznikají voláním funkce `execve()` a **jejich počátek se následně nikdy nepohybuje**. Mezi ně patří text, data, halda, kernelspace-stack a userspace-stack.
- *Mapování vzniklá voláním* `mmap()` jsou mapování, která mohou vznikat a zanikat kdykoliv během existence procesu.

Všechna mapování obecně mohou měnit svou velikost během běhu procesu.

4.6.1 Kód programu (text)

Mapování text je umisťováno zpravidla na nejnižší adresy AS procesu. V tomto mapování jsou uloženy

- **instrukce programu**,

Posloupnost strojových instrukcí implementujících funkce zapsané přímo ve zdrojovém kódu programu.

- **instrukce staticky linkovaných knihoven** a

  Posloupnost strojových instrukcí implementujících funkce ze staticky linkovaných knihoven, které program volá.

- **konstanty**.

  Neboť hodnoty konstant překladač umisťuje rovnou do strojových instrukcí.

Obsah tohoto mapování je vykonáván procesorem. Pro fungování programu je nezbytné, aby bylo pro stránky, na nichž tato oblast leží, **povoleno čtení a vykonávání** jejich obsahu.

4.6.2 Oblast inicializovaných dat (data)

Mapování data je umisťováno zpravidla za oblast text. Obsahuje

statické proměnné inicializované ve zdrojovém kódu programu.
Proměnné jejichž existence není podmíněna přítomností *aktivačního záznamu rozsahové jednotky* (viz kapitola 4.6.7.3) na zásobníku. Svou hodnotu si pamatují po celou dobu běhu programu. Typicky se jedná o
globální proměnné a
Proměnné deklarované mimo jakoukoliv rozsahovou jednotku.
lokální „static“ proměnné.
Proměnné deklarované v rámci rozsahové jednotky, jejichž deklarace zahrnuje klíčové slovo `static`.

Umístění dat do této oblasti eliminuje potřebu alokace prostoru za běhu programu na haldě či na zásobníku.

4.6.3 Oblast neinicializovaných dat (BSS)

Mapování BSS je umisťováno za oblast data. Obsahuje data téhož druhu jako oblast data, od které se odlišuje

- tím, že obsahuje statické proměnné, které **nejsou** ve zdrojovém kódu programu inicializovány a
- inicializací všech bitů při startu programu hodnotou nula.

Výše uvedené koresponduje s faktem, že neinicializované `static` lokální proměnné a proměnné globální jsou v programu vždy inicializovány hodnotou nula.

4.6.4 Halda (halda)

Mapování halda je mapování, do něhož se umisťují proměnné, kterým nemůže být přidělen prostor již při překladu. Jde zejména o

- posloupnosti instrukcí dynamicky linkovaných knihoven, jejichž funkce jsou volány programem
- a data dynamicky[8] alokovaných proměnných.

4.6.5 Oblast sdílených knihoven (MMS)

Mapování MMS je mapování, do kterého se mapují soubory obsahující strojový kód dynamicky linkovaných knihoven. Pokud vykonávaný proces nevyužívá při svém běhu kód žádné dynamicky linkované knihovny, pak oblast MMS v jeho VAS přítomna není. Ve VAS leží mezi zásobníkem a haldou.

4.6.6 Zásobník (kernelspace-stack)

Ke každému vláknu v User Space existuje v Kernel Space právě jeden kernelspace-stack do nějž jsou ukládány záznamy o systémových voláních týkajících se vykonávání daného vlákna.

4.6.7 Zásobník (userspace-stack)

Zásobník (userspace-stack) je mapování umisťované do AS procesu a to v takovém počtu, kolik má proces vláken, čili každé vlákno má zásobník vlastní. Jedná se o anonymní mapování, což znamená, že pro něj neexistuje předloha v rámci FS, nýbrž vzniká vždy prázdný při volání funkce `execve()` (při vytváření procesu). Zásobník lze označit za jednosměrný spojový seznam tzv. *aktivačních záznamů (AZ)* (viz kapitola 4.6.7.3) reprezentujících tzv. *rozsahové jednotky (RJ)*. Před pokračováním v dalším textu je vhodné se seznámit s významem zkratek EBP, ESP, EIP, RIP a SFP v tabulce 4.2.

Programy psané ve vysokoúrovňových jazycích jsou strukturovány do procedur a funkcí, které se vzájemně volají. Díky tomu lze dekomponovat řešení problému do menších celků, kdy každá funkce může představovat prostředek pro řešení části z celkového problému. Každá jedna funkce může disponovat vlastními prostředky (např. proměnný-

[8] Funkcemi `malloc()`, `calloc()` a `realloc()` alokovaných proměnných.

Zkratka	Význam
EBP	*Extended Base Pointer* je registr x86-kompatibilního procesoru obsahující adresu počátku aktivačního záznamu odpovídajícímu aktuálně vykonávané rozsahové jednotce.
ESP	*Extended Stack Pointer* je registr x86-kompatibilního procesoru obsahující adresu vrcholu zásobníku.
EIP	*Extended Instruction Pointer* je x86-kompatibilního procesoru obsahující adresu právě vykonávané instrukce.
RIP	Adresa paměti, na níž leží instrukce, která se bude vykonávat po opuštění rozsahové jednotky, k níž je aktivační záznam vázán.
SFP	Ukazatel na začátek nadřazeného AZ.
OFP	Předchozí (1 krok starý) ukazatel na začátek nadřazeného AZ.

Tabulka 4.2: Vysvětlení významu používaných zkratek

mi). Informace o těchto, pro danou funkci exkluzivních, prostředcích se ukládá v rámci AZ, což je souvislý paměťový prostor v userspace-stack.

Při vstupu vykonávání programu do rozsahové jednotky vzniknuvší aktivační záznam je zařazen na vrchol zásobníku. Záznam na vrcholu zásobníku reprezentuje rozsahovou jednotku, v jejímž rámci (adresním prostoru) se nachází vykonávání programu (čítač instrukcí).

4.6.7.1 Příčiny volby zásobníku pro úschovu AZ

Datová struktura zásobník byla pro uchování AZ zvolena z následujících příčin:

- **Programy vyžadují podporu pro rekurzivní volání procedur.**

  Při volání podprogramů vzniká hierarchická struktura, kdy vždy jeden podprogram je v roli žadatele o řešení problému a druhý v roli řešitele problému. Zásobník umožňuje žádat o řešení problému tentýž podprogram, neboť zachovává historii podání žádostí.
- **Zásobník udržuje informaci o aktuálním stavu vykonávání programu.**

  Samozřejmě ji neudržuje celou – podstatná část informace o stavu vykonávání programu je uložena v registrech procesoru.

4.6.7.2 Rozsahová jednotka programu

Programový kód je možné členit na úseky nazývané *rozsahové jednotky* jako jsou např. blok[9], procedura nebo funkce. Rozsahová jednotka sestává z posloupnosti strojových instrukcí jí tvořících a odpovídá jí proto část AS procesu, na které dané instrukce leží. Vykonává-li se rozsahová jednotka, vykonává se některá z jejích instrukcí. První rozsahovou jednotkou v níž se hlavní vlákno procesu vykonávající program ocitne je funkce `main()`. Z ní pak může vstupovat do dalších rozsahových jednotek. Při vstupu vykonávání programu do rozsahové jednotky vzniknuvší aktivační záznam je zařazen na vrchol US zásobníku. Záznam na vrcholu zásobníku reprezentuje rozsahovou jednotku, v jejímž rámci (adresním prostoru) se nachází vykonávání programu.

4.6.7.3 Aktivační záznam

Aktivační záznam (AZ) je datová struktura, která představuje lokální prostředí výpočtu – tzv. kontext – v němž se nachází vlákno při vstupu do rozsahové jednotky. Právě vstup vykonávání vlákna do rozsahové jednotky iniciuje alokaci aktivačního záznamu na vrcholu zásobníku náležícímu danému vláknu. Obsah AZ shrnuje tabulka 4.3.

4.7 Princip volání podprogramů

Vykonávání podprogramu lze rozdělit do 4 částí: volání, prolog, samotné vykonávání a epilog. Vyskytnou-li se v rámci jejich následujícího popisu konkrétní hodnoty, vztahují se k programu uvedenému v C kódu 4.1. Jeho podoby v x86 strojovém kódu jsou uvedeny v 4.1. Postup bude vysvětlován na i386-kompatibilní variantě. Pozice v programu je definována obsahem registrů procesoru a obsahem userspace-stack. Pro zjednodušení možno uvažovat pouze registry EBP, ESP a EIP.

4.7.1 Volání funkce

Volání funkce či obecněji *vstup do rozsahové jednotky* v C kódu 4.1 reprezentované voláním funkce `fn(a,2)` sestává z následujících sekvence kroků:

[9] V jazyce C blok rozsahovou jednotku netvoří.

Nosič	Obsah	Komentář
předchozí AZ	vstupní parametry	Hodnoty předané při volání funkce. Jsou přítomny samozřejmě pouze v případě, že rozsahovou jednotkou je podprogram či funkce.
AZ	návratová adresa NA	Na architektuře x86 se jedná o **první položku v rámci AZ**. Možnost přepsání NA je obecně považována za zranitelnost [Zah11]. Jedná se o dočasné uložení hodnoty registru EIP na zásobníku. Hodnota označována též jako *RIP*.
	ukazatel na předchozí (nadřazený) AZ	Jedná se o dočasné uložení hodnoty registru EBP na zásobníku. Hodnota označována též jako *SFP*.
	lokální proměnné	Proměnné alokované staticky v rámci rozsahové jednotky.
	vstupní parametry	Hodnoty předané při volání funkce.

Tabulka 4.3: Komponenty obsažené v aktivačním záznamu

```c
void fn (int d, int e)
{
    int     b[2];
    char    c[7] = "1234567";

    b[1] = e;
}

int main (void)
{
    int     a = 1;

    fn(a,2);
    return 0;
}
```

C kód 4.1: Zdrojový kód programu, na němž se kapitola 4.7 pokouší vysvětlit princip volání podprogramů. Program je přítomen na přiloženém CD v souboru `/root/examples/subroutine-calls-example.c`.

```
fn:
    pushl   %ebp
    movl    %esp, %ebp
    subl    $16, %esp
    movl    $875770417, -15(%ebp)
    movw    $13877, -11(%ebp)
    movb    $55, -9(%ebp)
    movl    12(%ebp), %eax
    movl    %eax, -4(%ebp)
    leave
    ret
main:
    pushl   %ebp
    movl    %esp, %ebp
    subl    $16, %esp
    movl    $1, -4(%ebp)
    pushl   $2
    pushl   -4(%ebp)
    call    fn
    addl    $8, %esp
    movl    $0, %eax
    leave
    ret
```

```
fn:
    pushq   %rbp
    movq    %rsp, %rbp
    movl    %edi, -36(%rbp)
    movl    %esi, -40(%rbp)
    movl    $875770417, -32(%rbp)
    movw    $13877, -28(%rbp)
    movb    $55, -26(%rbp)
    movl    -40(%rbp), %eax
    movl    %eax, -12(%rbp)
    popq    %rbp
    ret
main:
    pushq   %rbp
    movq    %rsp, %rbp
    subq    $16, %rsp
    movl    $1, -4(%rbp)
    movl    -4(%rbp), %eax
    movl    $2, %esi
    movl    %eax, %edi
    call    fn
    movl    $0, %eax
    leave
    ret
```

Strojový kód x86 4.1: Strojový kód kompatibilní s procesory x86 (vlevo) a x86-64 (vpravo)

4.7.1.1 Uložení parametrů předávaných funkci

Parametry předávané funkci se ukládají od konce. Před voláním funkce `fn(a,2)` a při podobě zásobníku ...

Adresa		Obsah	Komentář
	$x+?$	???	Počátek nadřazeného AZ.
	⋮	⋮	⋮
EBP →	$x-0$	$x+?$	Návratová adresa, kde bude vykonávání pokračovat po ukončení činnosti v aktuální RJ.
ESP →	$x-1$	VAR a	V aktuálním AZ lokální proměnná.
	$x-2$		

...se vykonají následující strojové instrukce, které tak připraví předání parametrů volané funkci skrze zásobník:

```
pushl $2
pushl $-4(%ebp)
```

Příprava parametrů na zásobníku tedy není automatizována v rámci instrukce `call` a procedura si ji musí vykonat sama. Vkládání na zásobník automaticky snižuje hodnotu uloženou v registru ESP a proto není potřeba pro předání parametrů dělat nic dalšího. Nová podoba zásobníku je následující:

Adresa		Obsah	Komentář
	$x+?$	???	Počátek nadřazeného AZ čili OFP.
	⋮	⋮	⋮
EBP →	$x-0$	$x+?$	Návratová adresa, kde bude vykonávání pokračovat po ukončení činnosti v aktuální RJ.
	$x-1$	VAR a	V aktuálním AZ lokální proměnná.
	$x-2$	2	Poslední parametr předávaný funkci.
ESP →	$x-3$	1	Předposlední parametr předávaný funkci.
	$x-4$		

4.7.1.2 Uložení návratové adresy na zásobník a skok na adresu uloženou v operandu instrukce `CALL`

Instrukce

```
call <fn>
```

způsobí

- uložení návratové adresy – hodnoty registru EIP zvýšené o 5 B na zásobník (tedy i snížení hodnoty v ESP) a
- následnou změnu hodnoty registru EIP na hodnotu, která byla instrukci `call` předána jako operand.

Adresa		Obsah	Komentář
	$x+?$	???	Počátek nadřazeného AZ.
	⋮	⋮	⋮
EBP →	$x-0$	$x+?$	Návratová adresa, kde bude vykonávání pokračovat po ukončení činnosti v aktuální RJ.
	$x-1$	VAR a	V aktuálním AZ lokální proměnná.
	$x-2$	2	Poslední parametr předávaný funkci.
	$x-3$	1	Předposlední parametr předávaný funkci.
ESP →	$x-4$	EIP+5	Návratová adresa čili RIP, na kterou se program vrátí po opuštění právě zakládaného AZ.
	$x-5$		

4.7.2 Prolog funkce

Prolog sestává z následujících operací:

4.7.2.1 Vytvoření nového AZ

Uložení hodnoty v EBP – SFP – na zásobník ...

```
push %ebp
```

...a nastavení hodnoty v EBP na na hodnotu registru ESP ...

```
movl %esp, %ebp
```

Adresa	Obsah	Komentář
$x+?$	???	Počátek nadřazeného AZ.
⋮	⋮	⋮
$x-0$	$x+?$	Návratová adresa, kde bude vykonávání pokračovat po ukončení činnosti v aktuální RJ.
$x-1$	VAR a	V aktuálním AZ lokální proměnná.
$x-2$	2	Poslední parametr předávaný funkci.
$x-3$	1	Předposlední parametr předávaný funkci.
$x-4$	EIP+5	Návratová adresa čili RIP, na kterou se program vrátí po opuštění právě zakládaného AZ.
EBP, ESP → $x-5$	EBP	Ukazatel na začátek předchozího aktivačního záznamu – SFP.
$x-6$		

4.7.2.2 Vyhrazení prostoru pro lokální proměnné

Zásobník v rámci x86 architektury pracuje s úseky délky 32 b, čili 4 B. Prostor vyhrazený pro lokální proměnné se proto zaokrouhluje. Registr ESP se sníží[10] o velikost místa vyhrazeného pro lokální proměnné.

```
subl $16,%esp
```

Adresa	Obsah	Komentář
$x+?$	???	Počátek nadřazeného AZ.
⋮	⋮	⋮
$x-0$	$x+?$	Návratová adresa, kde bude vykonávání pokračovat po ukončení činnosti v aktuální RJ.
$x-1$	VAR a	V aktuálním AZ lokální proměnná.
$x-2$	2	Poslední parametr předávaný funkci.
$x-3$	1	Předposlední parametr předávaný funkci.
$x-4$	EIP+5	Návratová adresa čili RIP, na kterou se program vrátí po opuštění právě zakládaného AZ.
EBP → $x-5$	EBP	Ukazatel na začátek předchozího aktivačního záznamu – SFP.
$x-6$	VAR b[1]	
$x-7$	VAR b[0]	
$x-8$	VAR c[6],c[5],c[4]	
ESP → $x-9$	VAR c[3],c[2],c[1],c[0]	
$x-10$		

4.7.3 Epilog funkce

4.7.3.1 Vrácení zásobníku do stavu totožného se stavem předcházejícímu volání funkce

Epilog funkce představuje ve strojovém kódu 4.1 dvojice instrukcí

```
leave
ret
```

Instrukce `leave` je zkratkou pro

```
movl %ebp, %esp
popl %ebp
```

[10] Sníží, neboť zásobník „roste“ směrem k od vyšších k nižším adresám.

. První do registru ESP přiřadí hodnotu uloženou v registru EBP. Druhá následně do registru EBP načte hodnotu z nového vrcholu zásobníku.

4.7.3.2 Nastavení návratové adresy

Instrukce `ret` je zkratkou pro

```
popl %eip
```

čili vyzvedne ze zásobníku NA (návratovou adresu) a uloží ji do registru EIP. Vyzvednutí čehokoliv ze zásobníku znamená též zvýšení hodnoty registru ESP. Po vykonání instrukce `ret` se stav zásobníku vrátí do podoby, jaká je vyobrazena v kapitole 4.7.1.1.

4.7.3.3 Odstranění parametrů ze zásobníku

Odstranění parametrů je prováděno již v kontextu funkce, která prováděla volání podprogramu (zde funkce `main()`). Jde o zvýšení hodnoty v registru ESP o počet bytů, které byly vyhrazeny parametrům:

```
addl $0x8, %esp
```

5. Standardní bezpečnostní řešení

Kapitola je věnována seznámení s vybranými bezpečnostními konstrukty, kterými Linux buďto disponuje ve své základní podobě nebo po aplikaci volně dostupných patchů.

5.1 Bezpečnostní mechanismy implementované v hardware

Bezpečnost nelze zajišťovat čistě softwarovou cestou. Hardware pro ni musí poskytovat podporu.

5.1.1 Rings

Základní bezpečnostní technologií pro systém, bez níž by nebylo možné rozvíjet další bezpečnostní technologie[1] další, jsou tzv. *rings*. Na x86-kompatibilních procesorech se jedná se o hierarchii 4 úrovní oprávnění, kde každá úroveň zahrnuje všechna oprávnění vzhledem k ní nižších úrovní. I zde se aplikuje známý „princip nejmenšího oprávnění“ [Den], čili kód by měl být vykonáván vždy jen s takovými oprávněními, jaká nezbytně potřebuje ke svému běhu. V rámci architektury x86 jsou rozlišovány

- **úroveň oprávnění Ring 0**,

  Nejvyšší možná úroveň oprávnění v rámci x86-kompatibilního procesoru zaručující neomezený přístup ke všem zařízením. V této úrovni oprávnění běží jádro operačního systému.
- **úroveň oprávnění Ring 1**,

  Druhá nejvyšší úroveň oprávnění v rámci x86-kompatibilního procesoru. V této úrovni oprávnění běží na systémech s Linuxem buďto hypervizor nebo nic.
- **úroveň oprávnění Ring 2** a

  Druhá nejnižší úroveň oprávnění v rámci x86-kompatibilního procesoru. Tato úroveň oprávnění není při nasazení GNU Linux využívána.
- **úroveň oprávnění Ring 3**.

  Nejnižší úroveň oprávnění v rámci x86-kompatibilního procesoru. V této úrovni běží všechny procesy běžící v systému. Kódu v této úrovni oprávnění není dovoleno přistupovat k paměťovým stránkám jiných AS nežli je jeho vlastní AS čili kód běžící v Ring 3 je vězněm vlastního AS a jakékoliv interakce s zařízeními či AS jiných procesů provádí pomocí jádra.

Přístupová oprávnění jednotlivých úrovní jsou dozorována MMU, který v případě pokusu o neoprávněný přístup do paměti, zjedná typicky výjimku Page fault, kterou následně vyhodnotí jádro zabitím daného procesu. Přepnutí do kernel mode (Ring 3 → Ring 0 → Ring 3) **je relativně drahou operací**, která spotřebuje kolem 1000-1500 cyklů. Pro porovnání s ní např. přepnutí kontextu (Ring 0 → Ring 0) spotřebovává cca 100 cyklů.

5.1.2 NX bit

No eXecute bit je bit přítomný v tabulce stránek nesoucí informaci, že obsah ležící v dané stránce není dovoleno spouštět. Díky němu Memory Management Unit může řídit pokusy o spouštění obsahu paměťových stránek (viz kapitola 4.2.2). Bohužel jeho přítomnost není Linuxem doposud příliš efektivně využívána (viz kapitola 9.2). **Podpora nespustitelných stránek paměti je v linuxu existuje standardně jen pro anonymní mapování. Její fungování a její možné rozšíření popisuje kapitola 9.2.** Za účelem lepšího využití NX bitu vznikl projekt PaX (viz kapitola 9). Na architekturách, kde není přítomen – typicky na x86-32 – je možné jej emulovat (viz kapitola 9.2.2), nicméně emulace nikdy nemůže dosáhnout kvalit nativní podpory implementované v hardware a přináší s sebou jistá omezení (viz kapitola 9.3).

5.2 Řízení přístupu

Nadefinování řízení přístupu vyžaduje znalost následujících termínů používaných v oblasti bezpečnosti:

uživatel
: Identita existující v rámci systému.

objekt
: Pasivní entita, se kterou může být manipulováno subjekty. Manipulace s ní je řízena (omezována). Pod tímto pojmem je možné si představit například
  - soubor (v širším slova pojetí),
  - socket,

- proces či
- prostředek meziprocesové komunikace.

subjekt
: Aktivní entita provádějící operace nad objektem. V prostředí OS se jedná **výhradně** o **proces**.
Subjekt může být pro jiný subjekt objektem, což platí např. při meziprocesové komunikaci, kdy adresát zprávy – proces – představuje v daném případě objekt.

operace
: Aktivita vyvolaná subjektem nad objektem.

povolení
: Autorizací získaný souhlas pro provedení operace.

Zajistit absolutní odolnost programu vůči veškerým existujícím hrozbám **je drahé**, má-li daný program něco užitečného dělat. **U naprosté většiny programů** až na čestné výjimky **jejich tvůrci** (i uživatelé) **šetří**, respektive se spokojí s pouze částečnou odolností. Vznikají tak z bezpečnostního hlediska nedostatečně dobře psané programy, což musí zákonitě vést ke zvýšené pravděpodobnosti neoprávněného převzetí kontroly nad během jejich instancí a tím i nabytí přístupových oprávnění vztažených k jejich efektivní identitě (EI). Páchání škod však nemusí být nutně pouze doménou subjektů, jejichž oprávnění byla získána zcizením identity. Škody mohou páchat též řádně autentizované subjekty, které nemusí mít nutně dobré úmysly či dostatečnou způsobilost k nakládání s objekty. Za účelem minimalizace škod je zapotřebí, aby byl přístup k objektům v systému omezován, respektive řízen.

Řízení přístupu (AC) je proces ověřující míru oprávnění subjektu k manipulaci s objektem. Běžně se rozlišují tři základní typy řízení přístupu:

- volitelné řízení přístupu (DAC),
- povinné řízení přístupu (MAC) a
- řízení přístupu založené na rolích (RBAC),

jejichž popis následuje.

5.3 Volitelné řízení přístupu (DAC)

Volitelné řízení přístupu (DAC) je řízení přístupu, kde **přístupová oprávnění k objektu definuje vlastník objektu**. Přístupová oprávnění k objektu může mimo jeho vlastníka modifikovat též privilegovaný uživatel, nicméně vlastník objektu může jeho vůli kdykoliv změnit.

5.3.1 Dopady na systém

Nasazení výhradně DAC implikuje následující skutečnosti:

Správce systému má omezené pravomoci.
: Správce nemá kontrolu nad přístupovými právy k objektům, jichž není vlastníkem.

Přístupová oprávnění jsou vázána výhradně na EI subjektu.
: Subjekt má
  - neomezená oprávnění pro manipulaci s objekty vlastněnými uživatelem s identitou rovnou EI procesu a
  - oprávnění pro manipulaci s objekty, jejichž vlastníci tato oprávnění delegovali na uživatele s identitou rovnou EI procesu.

Vůči znepřátelenému subjektu mohou být chráněny pouze některé objekty.
: Ten, kdo převezme kontrolu nad během procesu, se zmocní též všech přístupových oprávnění držených identitou, kterou má proces jako svou EI.
Vůči znepřátelenému subjektu nejsou nijak chráněny objekty, které vlastní uživatel, jehož identita je rovna EI procesu.

5.3.2 Implementace

Volitelné řízení přístupu (DAC) se realizuje prostřednictvím ACL.

Access Control List (ACL) je množinou uspořádaných dvojic

$$(\texttt{<množina-přístupových-oprávnění-k-objektu>}, \texttt{<množina-identit-subjektů>})$$

mezi nimiž jsou vždy přítomny dvojice, kde `<množinu-identit-subjektů>` tvoří

vlastník,
: Jednoprvková množina obsahující identitu subjektu vlastnícího daný objekt.

skupinový vlastník a
: Množina sestávající z identit subjektů, které sdílí s vlastnickým subjektem jeho primární skupinu.

ostatní.
: Množina identit subjektů v předchozích dvou množinách se nevyskytujících.

V operačních systémech GNU Linux jsou ACL realizujícími DAC

unixová přístupová práva a
: ACL umožňující specifikovat přístupová práva vlastníka, skupinového vlastníka a ostatních identit subjektů.

POSIX ACL.
: Nadmnožina unixových přístupových práv díky níž je možné definovat přístupová práva konečné množině množin subjektů.

Implementace volitelného řízení přístupu (DAC) jsou tedy realizovány jako seznamy – ACL. Každý objekt v rámci souborového systému disponuje svým vlastním ACL obsahujícím definice pravidel pro řízení přístupu vzhledem k různým množinám subjektů. Veškerá pravidla jsou ukládány do inode reprezentujícího daný objekt v souborovém systému. Upravovat je může pouze subjekt s identitou uživatele, jež je vlastníkem příslušného souboru.

5.3.3 Unixová přístupová práva

Ne/udělení oprávnění procesu k manipulaci se souborem závisí

na efektivní identitě (EI) procesu
: EI procesu je dána EUID a EGID.

a na definici přístupových oprávnění k přistupovanému souboru.
: Každý objekt v systému souborů má ve svém i-nodu uloženu EI procesu, který daný soubor vytvořil respektive UID uživatele (vlastníka souboru) a GID skupiny (skupinového vlastníka souboru). Od těchto hodnot se odvíjí klasifikace procesu/subjektu do tří disjunktních množin:

procesy vlastníka
: Všechny procesy s EUID rovným UID uživatele, jež soubor vytvořil či kterému byl soubor přenechán např. příkazem `chown`.

procesy skupinového vlastníka
: Všechny procesy s EUID různým od UID vlastníka a zároveň s EGID rovným GID skupinového vlastníka, kterýžto soubor vytvořila či jí byl přenechán např. příkazem `chgrp`.

ostatní procesy
: Procesy nepatřící do předešlých dvou množin.

Pro každou množinu procesů jsou stanovena přístupová práva k souboru třemi příznaky, jež má každý soubor stejně tak, jako je tomu u UID a GID vlastníka, uloženy ve svém i-uzlu. Význam jednotlivých příznaků zobrazuje tabulka 5.1. Proces vyhodnocení oprávnění procesu (klasifikace procesu) přístupu k souboru je zobrazen na obrázku 5.1.

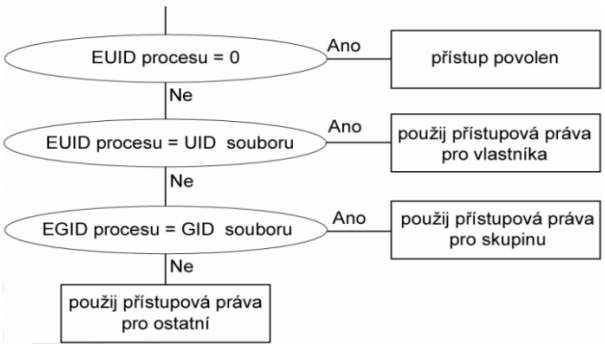
Příznak	Význam příznaku u souboru	Význam příznaku u adresáře
`r`	právo číst jeho obsah	právo vypsat v něm obsažené položky
`w`	právo modifikovat jeho obsah	právo modifikovat (vytvářet, přejmenovávat či mazat) v něm obsažené položky
`x`	právo spouštět jeho obsah	právo manipulovat s jednotlivými v adresáři obsaženými položkami (dle přístupových práv u nich nastavených), vstup do adresáře

Tabulka 5.1: Příznaky definující přístupová práva

5.3.3.1 Parametry nově vytvářených souborů

Z předchozího textu je zřejmé, že modifikovat přístupová oprávnění k objektu v případě DAC smí výhradně subjekt s EI vlastníka objektu či superuživatele. Nedefinuje-li subjekt explicitně přístupová práva při vytváření nového souboru, potom se tyto odvodí z jeho EI a tzv. implicitní masky práv. *Implicitní maska práv* je doplňkem do neomezených přístupových práv k objektu. Udržuje se pro tři množiny subjektů – vlastníka objektu, skupinového vlastníka objektu a ostatní subjekty. Subjekt může stanovit přístupová oprávnění jím vytvářeného objektu libovolně. Implicitní maska práv vlastnickým subjektem udělená přístupová oprávnění případně **redukuje**. Nastavit ji lze pomocí příkazu `umask` (viz příkaz 5.1).

Nově vytvářené soubory



Obrázek 5.1: Proces výběru oprávnění pro přístup k souboru

- patří uživateli, jehož UID je rovno EUID procesu, který daný soubor vytvořil, v čase vytvoření souboru,
- patří skupině, jejíž GID je rovno EGID procesu, který daný soubor vytvořil, v čase vytvoření souboru
- mají nastavena přístupová práva dle volby procesu, který je vytváří ponížená o práva uvedená v implicitní masce práv.

```
› umask 000 # nechť jsou přístupová práva plně v režii procesu
› touch a
› umask 020 # soubor zapisovatelný ostatními nesmí vzniknout
› cp a b
› ls -l
-rw-rw-rw- 1 miroslav miroslav 0 lis  1 16:49 a # proces v žádném případě nemusí nastavit všechna
 ↪  přípustná oprávnění
-rw-rw-r-- 1 miroslav miroslav 0 lis  1 16:50 b # v době vytvoření tohoto souboru byla práva pro zápis
 ↪  všem zakázána
```

Skript 5.1: Implicitní maska práv

5.3.3.2 Příznak setuid

Efektivní identita (EI) procesu je standardně tatáž, jako EI jeho rodiče. Pro proces jsou tedy standardně relevantní přístupová práva vztažená k identitě (uživateli), která jeho běh iniciovala. V tomto modelu nemá uživatel šanci provádět akce nad soubory jiných uživatelů bez příslušných oprávnění. Možnost **dočasného** přidělení přístupových oprávnění za účelem provedení **konkrétních akcí** nad souborem přináší příznak setuid (suid).

Setuid (suid) je příznak souboru, jehož nastavením se docílí toho, že EUID jakéhokoliv procesu vykonávajícího daný soubor bude rovno UID vlastníka daného souboru. Formálně je řazen mezi příznaky definující přístupová práva figurující ve volitelném řízení přístupu (DAC), nicméně na AC jako takové nemá žádný vliv.

5.3.3.3 Příklad využití setuid (suid)

Příkladem,

- ve kterém nestačí současné možnosti poskytované řízením přístupu (AC) založeném na ACL
- a který vyžaduje aplikaci mechanismu setuid (suid),

budiž udělení přístupových oprávnění subjektu **k části** objektu. Příčina nedostatečnosti AC pro řešení daného problému je zřejmá:

Rozlišovací schopnosti AC končí na úrovni jednotlivých souborů.
Řízení přístupu není dostatečně jemné, co se týče rozlišení jednotlivých objektů, k nimž je přístup řízen. Nejmenším rozlišitelným objektem v rámci AC k souborovému systému je soubor. Řídit přístup k jeho jednotlivým částem – např. znakům – není v současných implementacích AC možné.

AC aplikované na souborovém systému nejenže nedokáže řídit přístup k objektu menšímu nežli soubor. Dalším jeho úzkým místem je omezená množina AC rozpoznávaných akcí prováděných nad objektem. Řízení přístupu rozpoznává typicky pouze čtení, zápis a vykonávání obsahu objektu.

Řešení předchozích dvou nedostatků AC je možné zkonstruovat díky přítomnosti setuid (suid) příznaku:

1. Vytvořit program, který vykonává **právě a pouze povolenou činnost nad vybranými oblastmi** objektu, která je potřebná pro provedení požadovaných změn.

2. Nastavit identitu vlastníka programu rovnu identitě vlastníka objektu, v němž bude program provádět činnost.
3. Nastavit setuid (suid) bit na programu.

Tímto je dosaženo **minimalizace na subjekty delegovaných oprávnění** pro jejich nakládání s objektem. Toto řešení deleguje přístupová oprávnění na každý subjekt oprávněný ke spuštění programu s nastaveným setuid (suid) příznakem.

Pozorný čtenář v příkladu jistě vypozoroval zobecnění delegace práv v případě změny hesla používaného při autentizaci uživatele do systému, kde je výše zmíněné aplikováno:

1. Soubor obsahující hashe hesel – `/etc/shadow` – je ve vlastnictví superuživatele.
2. Neprivilegovaný uživatel nemá možnost změny svého hesla.
3. Je vytvořen program umožňující editaci jen a pouze hashe hesla uživatele, jehož identita je rovna RI procesu vykonávajícího tento program.
4. Daný program – `/usr/bin/passwd` – je ve vlastnictví téhož vlastníka, jakého má programem upravovaný soubor, respektive ve vlastnictví superuživatele.
5. Programu je nastaven setuid (suid) příznak.

Řešení využívající setuid (suid) bit tedy není zcela záležitostí jádra, neboť jeho část – instance setuid (suid) bitem obdařeného programu – je běžným uživatelským procesem, který je akorát vhodně „nakonfigurován“.

Při psaní programů, na nichž bude nastaven setuid (suid) bit, je vhodné se vyvarovat chyb. Tím spíše, budou-li prostřednictvím programu delegována přístupová oprávnění superuživatele.

5.3.3.4 Příznak setgid

Setgid (sgid) je příznak souboru, jehož nastavením se docílí toho, že EGID jakéhokoliv procesu vykonávajícího daný soubor bude rovno GID vlastníka daného souboru. Funguje analogicky jako příznak setuid (suid) uvedený v kapitole 5.3.3.2. V tomto případě je postihována skupinová komponenta EI procesu – EGID.

Nastavení setgid (sgid) příznaku na adresáři **má speciální význam**, a to ten, že skupinovým vlastníkem všech nově vytvořených souborů (v širším slova smyslu) v něm vytvořených bude tentýž skupinový vlastník, který vlastní daný adresář namísto toho, aby skupinovým vlastníkem nově vytvářených souborů byla skupina, jejíž GID má subjekt v době vytvoření souboru nastavenu jako EGID.

5.3.3.5 Sticky bit

Sticky bit je příznak, který je-li nastaven na adresáři, redukuje přístupová oprávnění k němu, konkrétně pak přístupová oprávnění zápisu jednotlivých subjektů. Subjektu, který je oprávněn modifikovat obsah adresáře, je toto oprávnění redukováno na oprávnění modifikovat výhradně obsah adresáře, jehož vlastníkem je uživatel s identitou rovnou EI subjektu.

Sticky bit je nastavován na adresářích, kde je zapotřebí zachovat oprávnění zápisu do adresáře alespoň dvěma uživatelům a zároveň odebrat oprávnění v adresáři mazat či přejmenovávat objekty ostatních uživatelů. Sticky bit je nastaven např. na adresáři `/tmp`, do nějž mají právo zapisovat všichni uživatelé v systému, avšak mazat a přejmenovávat si každý může pouze své soubory.

5.3.4 POSIX ACL

POSIX ACL jsou rozšířením základních přístupových práv. Umožňují definovat libovolně[2] dlouhé seznamy přístupových oprávnění, přičemž základní přístupová práva zůstávají vždy nezměněna. Manipulace prováděné s objektem se i v případě POSIX ACL stále rozlišují pouze na čtení, zápis a spuštění. Podpora POSIX ACL je dnes v systémových nástrojích široce zakomponována – viz interakce se shellem 5.2.

5.3.4.1 Nedostatky unixových přístupových práv řešené POSIX ACL

Rozlišení pouhých tří množin subjektů, které umožňují unixová přístupová práva zmíněná v kapitole 5.3.3, **je** v mnohých případech **nedostačující**. Projeví se to v případě

přidělení oprávnění množině uživatelů či

Při použití základních přístupových práv toto zadání znamená:

1. Vytvořit skupinu.
2. Přidat dotčené uživatele do této skupiny.
3. Změnit skupinového vlastníka souboru na nově vytvořenou skupinu.

[2] V praxi bývá počet záznamů omezen použitým souborovým systémem.

```
> touch a
> ls -l a
-rw------- 1 miroslav miroslav 0 říj  3 12:04 a  # ACL nevyužito
> getfacl a
# file: a
# owner: miroslav
# group: miroslav
user::rw-
group::---
other::---
> setfacl -m u:man:r a  # přidání práva číst soubor uživateli "man"
> ls -l a
-rw-r-----+ 1 miroslav miroslav 0 říj  4 08:56 a  # ACL využito (což signalizuje přítomnost symbolu
↪  '+' ve výstupu příkazu "ls")
> getfacl a
# file: a
# owner: miroslav
# group: miroslav
user::rw-
user:man:r--  # přístopová práva uživatele "man"
group::---
mask::r--
other::---
```

Skript 5.2: Zobrazení přístupových práv

4. Nastavit požadovaná oprávnění skupině.

   Toto není ve většině situací prakticky proveditelné, neboť neprivilegovaný uživatel standardně nemá možnost vytvářet skupiny, natož pak do nich přidávat uživatele. Absolvování celého tohoto procesu kvůli třeba i jednomu souboru lze označit jako „nepřiměřenou režii“. Se specifickými požadavky na řízení přístupu roste počet skupin za tímto účelem vytvářených. Již při mírně složitějších konfiguracích se toto řešení stává neudržovatelným.

přidělení oprávnění více než jedné skupině uživatelů.
: Tento čin v rámci unixových přístupových práv vyžaduje opět vytvoření speciální skupiny, která pojme veškeré uživatele z oprávnění hodných skupin.

5.4 Chroot

Chroot je systémové volání, které změní kořenový adresář procesu a všech jeho případných potomků vzniklých voláním `fork()` za účelem jejich izolace od zbytku systému. Jeho volání vyžaduje identitu privilegovaného uživatele. **Motivací** k izolaci procesu **je nedůvěra** v jeho budoucí činění, respektive nenulová pravděpodobnost toho, že program nebude vykonávat jen a pouze to, co deklaruje, že bude vykonávat. Vykonávání nedeklarovaného může být způsobeno

prvoplánově špatnými úmysly (autora) programu,
: Program vykonává činnost, které si je jeho autor vědom a tají ji.

chybou v programu nebo
: Program vykonává činnost, které si není jeho autor vědom nebo si jí vědom je, ale v tom případě jí netají.

dodatečně získanými špatnými úmysly programu.
: Program vykonává činnost, k níž jej pobízí útočník. Jedná se o případ, kdy útočník převezme kontrolu nad během programu. V podstatě je tato příčina vykonávání nedeklarovaného zapříčiněna chybou v programu. Samotná chyba v programu však představuje pouze zranitelnost, která se bez další aktivity (útočníka) hrozbou nestane.

Prostředí, do nějž je proces izolován se nazývá *chroot prostředí*. Toto prostředí je představováno

fiktivním kořenovým adresářem
: Uvězněný proces a jeho potomci vnímají kořenový adresář jim přiděleného podstromu jako kořen souborového systému. K souborům ležícím mimo chroot prostředí se izolované procesy nedostanou[1].

obsahujícím veškeré soubory potřebné pro běh uvězněných procesů,
: Do chroot prostředí je pro správný chod programu nutné umístit též soubory nezbytně nutné pro jeho běh

[1] Za předpokladu, že buďto nevyužijí žádné techniky umožňující neoprávněné opuštění chroot prostředí anebo sice využijí, ale jejich snaha bude zmařena díky aplikaci patche Grsec na použitém jádře. Metody, kterými Grsec posiluje bezpečnost chroot prostředí jsou popsány v kapitole 6.6.

včetně použitých sdílených knihoven.
To znamená duplikaci sdílených knihoven jak v externí paměti, tak i v paměti hlavní.

které nijak neomezuje meziprocesovou komunikaci.
Omezení chroot prostředí se vztahují pouze na (ne)viditelnost souborů. Uvězněné procesy vidí všechny neuvězněné procesy a naopak. Možnosti meziprocesové komunikace, jako jsou např. signály, **nejsou nijak omezeny**.

Pokud by z chroot prostředí nebylo možné uniknout, mohl by útočník škodit pouze uvnitř něj a přinejhorším spotřebovávat systémové zdroje, nicméně data mimo chroot prostředí by nepoškodil. Patch Grsec omezuje dále programy běžící v chrootu. Bez dodatečného zabezpečení však z chroot prostředí uniknout lze.

5.4.1 Jail

Jail je nástroj pro automatizaci vytváření chroot prostředí s mnohými vylepšeními.

Izolace procesu je při použití nástroje Jail důslednější, neboť dochází k oddělení nejen souborových systémů, ale i k oddělení

procesů a
Procesy izolované nástrojem jail vidí **pouze** procesy izolované v tomtéž prostředí.

síťových prostředků.
V každém izolovaném prostředí je možné přistupovat k síti **výhradně** prostřednictvím unikátní IP adresy svázané s daným prostředím.

Proces se superuživatelskou identitou bude na rozdíl od chroot prostředí v případě jail prostředí krácen na svých právech v oblastech

modifikace jádra,
Načítání a odebírání modulů jádra za běhu.

změny nastavení sítě,
Manipulace se síťovými zařízeními, modifikace adres, modifikace směrovacích tabulek, ...

připojování a odpojování souborových systémů,

změny parametrů jádra
Konfigurace voleb jádra příkazem `sysctl`.

a dalších ...

Takováto redukce oprávnění superuživatele vede k odolnosti proti únikům procesů z jail prostředí.

5.5 Capabilities

Capabilities je mechanismus řízení přístupu [16] zavedený v Linuxu 2.2. Oprávnění dříve exkluzivní pro superuživatele rozděluje do dále nedělitelných podmnožin oprávnění a ty umožňuje delegovat i na neprivilegované (obecně jakékoliv) uživatele, přesněji řečeno na procesy či dokonce jen vlákna těchto uživatelů. Informace o přístupových oprávněních k objektům je v případě capabilit nesena subjekty samými.

V systému bez capabilities nelze delegovat oprávnění subjektu nesoucího privilegovanou identitu na ostatní subjekty. Proto i pouhé naslouchání na privilegovaném portu vyžaduje superuživatelská oprávnění procesu. Zmocnění se procesu dává útočníkovi k dispozici identitu, na kterou nejsou aplikovány žádné systémové kontroly oprávnění a která tak může provádět cokoliv. Útoky jsou proto cíleny především na získání superuživatelské identity.

Díky capabilities je možné oprávnění superuživatele delegovat na subjekty běžící pod jinou (neprivilegovanou) identitou, čímž jsou zmírněny škody, které může napáchat útočník v případě získání kontroly nad během subjektu, neboť subjekt v případě využití capabilities nemusí běžet pod superuživatelskou identitou.

Capabilities lze procesu přidělovat za chodu. Různých capabilities existuje celá řada, například

`CAP_NET_BIND_SERVICE`,
Oprávnění procesu naslouchat na privilegovaném portu.

`CAP_CHOWN`,
Oprávnění procesu měnit vlastníka libovolného souboru.

`CAP_SYS_TIME`
Oprávnění procesu nastavit systémový čas.

a mnohé další (viz `man capabilities`).

5.5.1 Druhy capabilities

Capabilities každého subjektu (procesu) se dělí na

permitted (přidělené) a
: Množina všech capabilit, kterými proces disponuje. Přítomnost capability v této množině nemá vliv na řízení přístupu (AC).

effective (efektivní),
: Podmnožina[3] přidělených capabilit. Jsou používány při řízení přístupu subjektu k objektům.
Při pokusu o provedení privilegované operace jádro nezkoumá, zda-li je EI procesu superuživatelská, nýbrž zkoumá, zda-li je capabilita potřebná pro provedení této operace přítomna v množině efektivních capabilit.

inheritable (dědičné) capabilities.
: Množina obsahující capability, které budou přeneseny na proces, jež nahradí aktuální proces (voláním `execve()`). Volání funkce `fork()` jednotlivé množiny capabilities zachovává.

5.5.2 Získání capabilities

Získat capabilities lze třemi způsoby:

5.5.2.1 Systémovým voláním `capset()` za běhu procesu

1. Proces je spuštěn pod EI superuživatele. Typickým příkladem jsou programy s nastaveným SUID bitem (viz kapitola 5.3.3.2).
2. Proces zavolá `capset()`, čímž si přidělí jím požadované capability.
3. Proces se zbaví EI superuživatele voláním `seteuid()` či `setreuid()`.

5.5.2.2 Systémovým voláním `capsetp()` za běhu procesu

1. Proces není spuštěn pod EI superuživatele, respektive může být – ničemu to nevadí.
2. Jiný proces spuštěný pod EI superuživatele mu přidělí capabilities voláním `capsetp()`.

5.5.2.3 Nastavením atributu spustitelného souboru

Capabilities mohou být přiřazeny též spustitelnému souboru, respektive všem procesům v budoucnosti jej provádějícím. Informace o capabilities přiřazeným spustitelnému souboru se ukládají do inode tohoto souboru. Capabilities přiděluje superuživatel příkazem `setcap`.

5.5.3 Linux Intrusion Detection System

Linux Intrusion Detection System (LIDS) je patch jádra doplněný o jím přidanou funkcionalitu konfigurující nástroje. Staví z velké části na capabilities. LIDS poskytuje možnost

chránit/skrývat soubory před přístupy/pohledy neprivilegovaných,
: Řízení přístupu (AC) založený na LIDS má tu specifickou vlastnost, že rozlišuje pouze dvě identity – privilegovanou a neprivilegovanou.

chránit/skrývat procesy před signály/pohledy ostatních,
: Nikdo včetně administrátora nemůže ukončit pomocí signálu `SIGKILL` chráněné procesy. Procesy mohou být neviditelné.

vytvářet ACL založené na capabilities či
: LIDS poskytuje příkazy usnadňující manipulaci s capabilities, jejichž AC rozšiřuje. Například v případě capability `CAP_NET_BIND_SERVICE` přidává možnost podobou výčtu definovat porty, na které se daná capabilita vztahuje.

detekovat průnik do systému.
: *Intrusion Detection System (IDS)* – systém detekující neautorizovaný průnik do IS – implementovaný v rámci LIDS dokáže detekovat mnohé síťové útoky či jejich předvoj v podobě skenování otevřených portů.

Linux Intrusion Detection System (LIDS) se konfigurují příkazem `lidsconf` (viz příkazy 5.1 a 5.2. Nevýhodou LIDS je skutečnost, že neuvažuje různé uživatelské identity. Nastavení jím provedená se váží na všechny subjekty.

[3] Účelem koexistence množiny přidělených capabilit a množiny efektivních capabilit je možnost **dočasné deaktivace** capability procesu jejím odebráním z množiny efektivních capabilit.

```
lidsconf -A -s /bin/httpd -o CAP_NET_BIND_SERVICE 80-80,443-443 -j GRANT
```

Příkaz/y 5.1: Přidělení oprávnění k naslouchání na privilegovaných portech, na kterých může instance `/bin/httpd` naslouchat

```
lidsconf -A -o /etc/shadow -j DENY
lidsconf -A -s /bin/login -o /etc/shadow -j READONLY
```

Příkaz/y 5.2: Zákaz přístupu k `/etc/shadow` s výjimkou instancí `/bin/login`

6. Nadstandardní bezpečnostní řešení

6.1 Povinné řízení přístupu (MAC)

Povinné řízení přístupu (MAC) je řízení přístupu, kde **přístupová oprávnění k objektu definuje pověřená osoba – typicky správce systému**.

V případě přítomnosti definice pravidel MAC v systému je proces řízení přístupu dvoufázový, kde hlavní roli hrají definice přístupových oprávnění MAC:

1. fáze: dotaz vede do MAC-ACL
: Je-li přítomna definice přístupových oprávnění vážící se k právě vyhodnocovanému přístupu subjektu k objektu, potom je
  - MAC definice přístupových oprávnění v procesu autorizace rozhodující,
  - respektive DAC definice přístupových oprávnění nepodstatná.

2. fáze: dotaz vede do DAC-ACL
: Definice přístupových oprávnění v DAC je podstatná v případech, kdy v MAC pro přístup daného subjektu k danému objektu definice přístupových oprávnění schází. DAC na rozdíl od MAC obsahuje definici přístupových oprávnění **pro každou myslitelnou operaci**.

6.1.1 Dopady na systém

Nasazení MAC implikuje následující skutečnosti:

Vůči znepřátelenému subjektu mohou být chráněny všechny objekty.
: MAC umožňuje řídit (omezit) přístup subjektů též k objektům, jejichž vlastníkem je uživatel, jehož identita se rovná EI subjektu.

Správce systému má absolutní moc.
: Udělování přístupových oprávnění není v kompetenci vlastníka objektu, k němuž subjekt přistupuje.

6.2 Řízení přístupu založené na rolích (RBAC)

Řízení přístupu založené na rolích (RBAC) je MAC doplněné o hierarchii rolí. Mimo již nadefinované pojmy subjekt a objekt RBAC zavádí

transakce a
: Jakákoliv aktivita v systému vyvolaná subjektem s výjimkou autentizace subjektu.

role.
: *Role* je povolením vykonávat množinu transakcí udělované subjektu správcem systému.
Role mohou tvořit hierarchickou strukturu, kde nabytí role potomka způsobí též nabytí všech jeho rodičovských rolí. Vzhledem k subjektu může role nabývat tří stavů:

  nepřiřazená role
  : Role, kterou subjekt nedisponuje. Transakce vyžadující danou roli nemohou být subjektem vyvolány.

  přiřazená role
  : Role, kterou subjekt disponuje. Subjekt může mít přiřazeno vícero rolí, kde každá z nich ho opravňuje k určité množině transakcí.

  vybraná (aktivní) role
  : Subjekt má vždy aktivní **nejvýše jednu** roli, která ho opravňuje vykonávat určitou množinu transakcí. Aktivovat lze pochopitelně pouze role z množiny subjektu přiřazených rolí.

6.2.1 Implementace

Implementace RBAC musí dodržovat následující tři pravidla:

- Subjekt může inicializovat transakci pouze tehdy, má-li přiřazenu a vybránu roli.
- Aktivní role subjektu musí být pro subjekt autorizována.
- Subjekt smí vykonávat transakci pouze tehdy, je-li daná transakce autorizována pro jeho **aktivní** roli.

Dnes známé implementace MAC lze dělit dle mnoha kritérií, z nichž nejvýznamnější jsou

typ pojítka mezi pravidly a objekty/subjekty a

Jsou rozlišovány implementace MAC

založené na cestě – tzv. path based MAC a

Objekty jsou identifikovány cestami k nim vedoucími z kořene FS.

Nevýhodou tohoto řešení je **možnost vzniku nekonzistencí** v AC. Přístup k souboru, k němuž vede více pevných odkazů, může být řízen různě v závislosti na tom, prostřednictvím kterého pevného odkazu je k souboru přistupováno. MAC totiž umožňuje definici různých přístupových oprávnění pro každý pevný odkaz, neboť se od sebe odlišují svými adresami v rámci FS. **Výhodou** tohoto řešení je **absence nutnosti údržby značek** odpovídajících jednotlivým souborům.

založené na značce – tzv. label based MAC.

Objekty jsou identifikovány značkami, což

vyžaduje jejich údržbu,

Diskontinuita v údržbě značek může vést ke vzniku nekonzistencí.

eliminuje problém s více pevnými odkazy směřujícími na tentýž soubor

Soubor je v rámci tohoto řešení reprezentován svým i-nodem, což je pro účely AC nepochybně přirozenější.

a přidává další vrstvu abstrakce.

Značku lze, na rozdíl od pojmenování souboru v rámci FS, sdílet, respektive přidělit více souborům zároveň a snížit tak v některých případech počet potřebných pravidel.

využití či nevyužití rozhraní definovaného LSM.

MAC využívající LSM

Navázaní MAC na jádro prostřednictvím standardizovaného rozhraní LSM

nadřazuje DAC nad MAC a

Zamítnutí přístupu ze strany DAC má vyšší prioritu než povolení přístupu v MAC (viz kapitola 6.3.3).

šetří námahu vývojářům MAC.

Napojení vlastního AC do jádra je přítomností standardizovaného rozhraní maximálně zjednodušeno.

MAC nevyužívající LSM

Implementace AC do jádra napojená bez prostředníka, respektive napojující se vlastními silami.

6.3 Linux Security Modules

Linux Security Modules (LSM) je patch jádra obohacující jej o rozhraní budované **za účelem usnadnění** implementace různých AC. Standardní součástí jádra je LSM od verze 2.6. Vyjma implementací DAC v podobě unixových přístupových práv a POSIX ACL jsou implementace AC (především pak MAC) do jádra dodávány prostřednictvím patchů. Implementace AC obvykle sledují stále tytéž funkce jádra a snaží se je nějakým způsobem modifikovat. Každá po svém. LSM jim k tomu dává příležitost, neboť do struktur jádra přidává ukazatele na funkce, které mohou být spuštěny po všech běžných kontrolách oprávnění, které jádro standardně provádí. Definice těchto funkcí mohou být implementovány právě bezpečnostními záplatami jako je např. SELinux. Pokud jádro přístup povolí, může jej ještě zamítnout jedna z funkcí implementujících LSM. **Opačně to nefunguje – LSM neumožňují dodatečně povolit něco, co jádro již jednou zakázalo.**

LSM

poskytuje standardizované rozhraní pro AC a

Realizátorovi AC namísto zásahů do jaderných funkcí stačí, aby využil poskytované rozhraní, což mu ušetří čas.

umožňuje výměnu implementací AC za běhu.

LSM je postaven tak, aby AC jej využívající bylo napsáno formou modulu jádra, který lze nahrát či odebrat z jádra za běhu.

Modifikace jádra patchem LSM [ElS]

- přidává nová pole pro uložení režijních informací LSM do datových struktur,
- přidává LSM háčky do funkcí,
- přidává nová systémová volání týkající se zabezpečení a
- přidává funkce, které umožní registraci i od-registraci LSM modulu k LSM háčku za běhu.

6.3.1 Modul

LSM modul je **funkce** implementovaná nasazeným AC, která je v různých situacích volána – vyzývána k zodpovězení otázky, zda povolit přístup. Volání modulů zajišťují LSM háčky. Nejedná se o upravenou jadernou funkci, nýbrž

o funkci zcela novou.

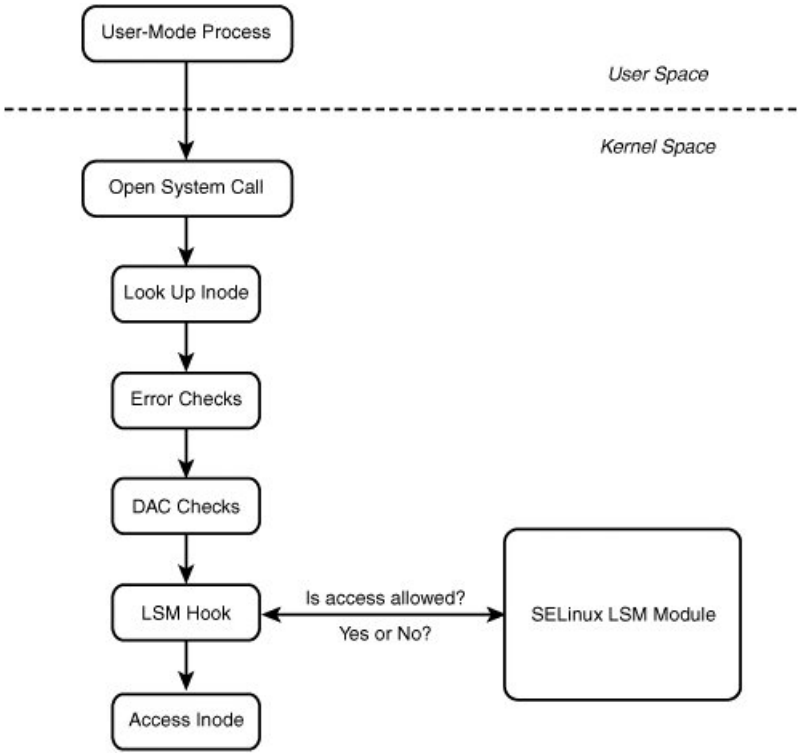
6.3.2 Háček

LSM hook je **volání funkce** – konkrétně LSM modulu.

LSM hooks jsou umisťovány do těl jaderných funkcí, ve kterých má smysl provádět AC. LSM hook nese adresu LSM modulu, pokud jej nasazené AC implementuje. Parametry předávané při volání LSM modulů jsou standardizovány.

Všechny LSM hooks jsou uloženy ve formě ukazatelů v globálně dostupné tabulce `security_ops`.

6.3.3 Princip AC při nasazení MAC postaveném na LSM



Obrázek 6.1: Schéma znázorňující zapojení LSM do procesu AC k souboru FS

Obrázek je převzat z [MMC07]

Zapojení LSM do procesu AC je vyobrazeno na obrázku 6.1, z něhož je patrné, že MAC využívající LSM je minimálně v otázce AC k souborům podřízené případnému přístup zamítajícímu rozhodnutí DAC. AC v případě nasazení MAC založeném na LSM aplikuje

nejprve stanovisko DAC
: DAC hraje roli primárního arbitra, co se týče zamezení přístupu subjektu k objektu. Jeho vůli v tomto směru nemá MAC šanci zvrátit.

a poté případně stanovisko MAC.
: MAC má možnost revokovat pouze přístup povolující rozhodnutí DAC.

6.3.4 Ukázka působení LSM v jádře

Příklad dokládající existenci LSM háčků v jaderných funkcích je zachycen v C kódu 6.1, kde

funkce `vfs_mkdir()`
: Funkce zprostředkovávající vytvoření nových adresářů ve FS.

obsahuje LSM hook realizující AC a
: LSM háček `security_ops->inode_ops->mkdir` je využit pro delegování rozhodovací pravomoci o udělení oprávnění k vytvoření nového adresáře na LSM modul – funkci, jejíž adresu háček obsahuje.
V případě nepřidělení oprávnění je návratová hodnota háčku, respektive hodnota proměnné `error` nenulová.

LSM hook pro manipulaci s položkami týkajícími se bezpečnosti.
: LSM háček `security_ops->inode_ops->post_mkdir` je použit pro nastavení položek v inode nově vytvořeného

```
int vfs_mkdir(struct inode *dir, struct dentry *dentry, int mode)
{
    int error;

    down(&dir->i_zombie);
    error = may_create(dir, dentry);

    if (error)
    {
        goto exit_lock;
    }

    error = -EPERM;

    if (!dir->i_op || !dir->i_op->mkdir)
    {
        goto exit_lock;
    }

    mode &= (S_IRWXUGO|S_ISVTX);

    error = security_ops->inode_ops->mkdir(dir, dentry, mode);

    if (error)
    {
        goto exit_lock;
    }

    DQUOT_INIT(dir);
    lock_kernel();

    error = dir->i_op->mkdir(dir, dentry, mode);
    unlock_kernel();

exit_lock:
    up(&dir->i_zombie);
    if (!error)
    {
        inode_dir_notify(dir, DN_CREATE);
        security_ops->inode_ops->post_mkdir(dir, dentry, mode);
    }
    return error;
}
```

C kód 6.1: LSM háčky v rámci funkce `vfs_mkdir()`

adresáře týkajících se řízení přístupu.

6.4 AppArmor

AppArmor je patch Linuxu postavený na LSM obohacující jej o vlastní implementaci MAC.

Rozdělení kompetencí mezi DAC a MAC odpovídá, vzhledem k využití LSM, principu popsanému v kapitole 6.3.3.

MAC je aplikováno **výhradně** na procesy vykonávající programy, pro které existuje definice přístupových oprávnění označovaná též jako AppArmor profil.

6.4.1 Profil

AppArmor profil je záznam v AppArmor-ACL určující přístupová oprávnění subjektu

- k socketům,
- k mechanismům meziprocesové komunikace (zasílání signálů) a
- k souborům (v širším slova pojetí).

Profily jsou umístěny v adresáři `/etc/apparmor.d`. Profil je svázán se subjektem prostřednictvím cesty k subjektem vykonávanému spustitelnému souboru v rámci souborového systému, z čehož plyne zařazení patche AppArmor mezi implementace MAC založené na cestách. AppArmor **nezná** role, tudíž neposkytuje možnost RBAC.

```
/usr/bin/apache
{
    network inet stream,    # program smí přistupovat k~síti
    /etc/ r,                # program smí číst /etc
    /bin/bash ixr,          # program smí číst+spustit /bin/bash (ale nedude se měnit profil nově
     ↪  spuštěného programu)
    deny /usr/lib/firefox-3.6.3/** w,   # zákaz zápisu do adresářů a podadresářů
    owner ${HOME}/.mozilla/** rwk,      # vlastník adresáře bude smět rw a zamykat soubry a adeesáře
}
```

Konfigurace 6.1: Příklad konfigurace MAC v podání AppArmor

6.5 SELinux

SELinux je patch jádra využívající rozhraní definovaná LSM, respektive jedná se o LSM modul. Jeho podpora je dnes přítomna jak ve vanilla jádře tak i ve většině distribučních jader. SELinux nemá learning mode, nicméně existuje externí nástroj generující nová pravidla na základě zamítnutých přístupů [Hol]. Je jím skript `audit2allow`.

Ověření oprávnění přístupu subjektu k objektu se řídí dle principu popsaného v kapitole 6.3.3, respektive i v případě SELinuxu platí, že zaujme-li DAC v otázce udělení přístupových oprávnění subjektu odmítavý postoj, potom celé AC končí neudělením přístupu.

6.5.1 Entity zavedené SELinuxem

Řízení přístupu (AC) definované patchem SELinux pracuje s množinou následujících pojmů:

6.5.1.1 Identita

SELinux identita je obdobou unixové identity – uživatele. Zavádí se pouze pro potřeby AC v rámci SELinuxu. Jedna SELinux identita může být přiřazena více uživatelům. Obecně je tak SELinux identita obdobou uživatelské skupiny, neboť má schopnost sdružovat uživatele. Je zvykem, že její pojmenování končí řetězcem `_u`.

Každá SELinux identita disponuje seznamem SELinux rolí do kterých se mohou subjekty, kterým je daná SELinux identita přiřazena, hlásit. Každý subjekt má přiřazenu SELinux identitu, která je během jeho běhu neměnná.

MAC poskytované SELinuxem samozřejmě může přístupová oprávnění členit již na úrovni identit. V řadě případů si však vystačí i s pouhopouhými dvěma SELinux identitami:

identitou `system_u` **a**
: Identita, pod kterou běží zpravidla systémem poskytované služby. Subjekty s touto identitou mají přístup k *SELinux roli* `system_r`.

identitou `user_u`.
: Identita, pod kterou běží subjekty ovládané běžnými uživateli, respektive neprivilegovanými uživateli. Subjekty této identity mají zpravidla možnost přihlašovat se do role `user_r`.

Pro rozlišení subjektů má SELinux MAC v rámci SELinux kontextu (viz kapitola 6.5.1.5) mnohé další příležitosti.

6.5.1.2 Role

SELinux role je **množina operací**, které smí subjekt konkrétního typu v ní běžící vykonávat nad objekty. Každé jednotlivé operaci odpovídá jí povolující SELinux pravidlo, jehož podoba je popsána v kapitole 6.5.4.

Jednotlivé identity se mohou hlásit do vybraných rolí. Mapování **povolených rolí** na identity je definováno v samostatných souborech ve formátu zobrazeném SELinux konfigurací 6.1. Je zvykem identifikátory rolí zakončovat řetězcem `_r`.

```
user <SELinux-identita> role { <SELinux-role-A> [<SELinux-role-B> ...] };
```

SELinux 6.1: Syntaxe mapování rolí na identity v podání SELinuxu

6.5.1.3 Typ

SELinux typ je **množina** subjektů či objektů [Hol]. Do typu jsou jednotlivé objekty zařazeny tím, že se daný typ zapíše do jejich *SELinux kontextu* – *značky*. Každý subjekt/objekt musí patřit **do právě jednoho** typu.

V rámci AC pak všechny objekty téhož typu tvoří skupinu, k níž jsou udělována přístupová oprávnění. Názvy typuu jsou zpravidla zakončeny řetězcem `_t`.

6.5.1.4 Doména

SELinux doména je označení pro množinu subjektuu patřících do SELinux typu.

6.5.1.5 Kontext

SELinux kontext označovaný též jako značka je **uspořádaná čtveřice**

- SELinux identita,
- SELinux role a
- SELinux typ.

SELinux ke svému chodu vyžaduje, aby každý objekt i každý subjekt disponoval svým vlastním SELinux kontextem – tzn. aby systém byl označkován, neboť právě a jen od obsahu značek se odvíjí veškerá rozhodnutí o (ne)udělení přístupu v rámci SELinux politik.

SELinux kontext je dědičný, respektive subjekt dědí kontext po svém předkovi. Změna kontextu je možná pouze v případě, že existují pravidla danému subjektu jí povolující.

6.5.2 Místo uložení kontextu

Značky jsou ukládány do *EFA* [Kor13].

Extended file attributes (EFA) – rozšířené atributy souboru – je **funkce souborového systému** poskytovaná pro uchování libovolných metadat souboru. Tato metadata

jsou uložena v rámci inode souboru,
: Souborové systémy poskytující možnost jejich ukládání je ukládají do inode souboru [Ale14], respektive do datových bloků tohoto souboru v případě, že se metadata nevejdou do inode, čímž je umenšována, i když nepatrně, maximální velikost popisovaného souboru.

nejsou přenosná mezi různými FS a
: Jelikož jsou EFA pouze nadstavbou poskytovanou FS, nejsou obecně přenosná mezi různými FS, ba co víc, ani mezi dvěma instancemi FS téhož typu, neboť standardní nástroje manipulující se soubory (`cp`, `mv`, ...) je nemusí brát v potaz.

mohou obsahovat libovolné informace.
: Informace vedené v rámci EFA jsou uspořádány do posloupnosti uspořádaných dvojic (klíč,hodnota) [Kuč11].

Pro značky – SELinux kontext jednotlivých souborů – platí veškerá omezení daná implementací EFA stejně tak jako i pro jakákoliv jiná EFA metadata.

6.5.3 Módy běhu

Jsou rozlišovány následující tři stavy SELinuxu [Háj15]:

enforcing
: Mód, ve kterém je SELinux plně aktivní, respektive
  - vynucuje SELinux politiky a
  - pokusy o neoprávněné přístupy zaznamenává

permissive
: Mód, ve kterém SELinux nevymáhá nastavené bezpečnostní politiky, nicméně stále zaznamenává neoprávněné přístupy k objektum.

vypnutý
: Přechodem do tohoto módu se ztratí kontinuita značek souborů, neboť během odstávky SELinuxu nejsou nikým udržovány. Před opětovným zapnutím SELinuxu je proto nutné SELinux kontext – značku – každého objektu sestavit znovu.

6.5.4 Pravidlo

SELinux pravidlo je **předpis**

specifikující množinu povolených operací,
: Pravidla v SELinuxu jsou **povolující**, respektive
  - SELinux implicitně odebírá subjektům přístupová oprávnění a
  - pravidla přístupová oprávnění vybraným subjektům opět navyšují.

které mohou subjekty určitého typu
: subjekty, na něž se pravidlo vztahuje, pojí shodný SELinux typ.

provádět nad objekty určitého typu.
: objekty, na něž se pravidlo vztahuje, pojí shodný SELinux typ.

Syntaxe pravidla, která je mimochodem vyobrazena v SELinux konfiguraci 6.2, může obsahovat položky jako

```
allow <typ-subjektu> <typ-objektu> : <třída> { <operace> [<operace> ...] };
```

SELinux 6.2: Syntaxe SELinux pravidla

`<typ-subjektu>`,
: Typ identifikující množinu subjektů, kterým pravidlo povolí vyjmenované operace.

`<typ-objektu>`,
: Typ identifikující množinu objektů, na kterých budou pro dané subjekty povoleny vyjmenované operace.

`<třída>` **a**
: Specifikace druhu objektu ve smyslu soubor, proces, file deskriptor, ...

`<operace>`.
: Operace běžné nad danou třídou objektů – zápis, čtení, zjišťování atributů ...

Pravidla se při spuštění SELinuxu překládají do binární podoby za účelem redukce zdržení při AC.

6.5.4.1 Příklad pravidel

```
allow iceweasel_t home_root_t : dir { search getattr read } ;
allow iceweasel_t user_home_dir_t dir { search getattr read } ;
allow iceweasel_t ld_so_t : file { read execute getattr };
```

SELinux 6.3: Povolení přístupu subjektum typu `iceweasel_t` k vybraným operacím nad objekty

V ukázce SELinux konfigurace 6.3 jsou uvedena pravidla udělující oprávnění všem subjektům typu `iceweasel_t`

- ke vstupu a vypsání obsahu adresářů typu `home_root_t`,
- ke vstupu a vypsání obsahu adresářů typu `user_home_dir_t` a

- ke čtení a spouštění souborů typu `ld_so_t`.

6.6 Grsecurity

Grsecurity (Grsec) je soubor modifikací jádra posilujících bezpečnost distribuovaný formou patche. Na rozdíl od jiných implementací MAC Grsec nevyužívá LSM [Hor11] a to je jedním z důvodů, proč se zatím nedostal do vanilla jádra, které je tak nutné pro zprovoznění Grsec modifikovat. Posílení bezpečnosti poskytovaná Grsec eliminují zranitelnosti z různých oblastí činnosti jádra. Podrobnější popis jednotlivých vylepšení Grsec je věnována samostatná kapitola 9.

6.6.1 Cena

Od zadání práce se Grsec v mnohém významně změnilo. Funkční doplnění[1], které možná mnohého čtenáře sledujícího vývoj na poli bezpečnosti napadne jako první, není však tou hlavní změnou. Je jí cena. **V čase zadání bylo zdarma i to nejnovější stabilní vydání**. Ve srovnání bezpečnostních řešení v kapitole 7 mělo Grsec tehdy ještě jasnou převahu. Od září 2015 **Grsecurity přestalo být dostupné zdarma ve své stabilní verzi** [Spe15]. Stabilní vydání bylo od té doby k dispozici výhradně platícím zákazníkům. Ostatní mohli používat testovací vydání Grsec. Důvodem ke zpoplatnění stabilní verze prý bylo porušování licence GPL a ochranné známky grsecurity®. To citelně otřáslo pozicí Grsec, nicméně stále zůstávala volně dostupná testovací verze. Při nevyužití těch úplně nejnovějších bezpečnostních technik, které se právě dostaly na světlo boží v rámci testovací verze, nebyl důvod mít obavy z jejího používání. Dne 26. 4. 2017 bylo vydáno prohlášení [Spe17], ve kterém tým stojící za Grsec přenechává údržbu soudobých Grsec volně dostupných (testovacích) záplat komunitě a oznamuje, že jsou poslední. Dále se tým prý bude soustředit již jen na platící zákazníky. **Posledním jádrem, které lze zdarma opatřit patchem Grsec, byť jen v jeho testovací variantě, je jádro verze 4.9**. Přestože Grsec pro tuto verzi byl již vydán, z oficiálních stránek byl již 26. 4. 2017 stažen. Existuje možnost, že již zveřejněné testovací verze Grsecurity převezme jiný tým a ujme se jejich údržby a přidávání podpory pro novější verze jader. Autor Grsec – Brad Spengler – však nehodlá tyto aktivity nijak podporovat [Krč17]. Vidina snadno čili zdarma nabytého nadstandardního zabezpečení serverů CIV ZČU se tak rázem rozplynula. Jsa soucitným se čtenářem, vedoucí práce – Michal Švamberg – pojal za potřebné poskytnout alespoň informaci o ceně bezpečnostního řešení, které dokument analyzuje a jehož dopad měří. Autorovi práce proto udělil pověření jednat jménem CIV ZČU ve věci podání žádosti o nabídku. Ke dni 17. 5. 2017 poskytl tým vyvíjející Grsec nabídku pro CIV ZČU, o kterou byl nedlouho předtím požádán. Přesněji řečeno poskytl nabídky celkem čtyři. Všechny uvažují nasazení záplaty Grsec na 100 serverech patřících CIV ZČU a jsou uvedeny v tabulce 6.1.

Cena/rok [USD]	Cena/rok [Kč][2]	Produkt/služba
12400	297600	vždy aktuální zdrojové kódy Grsec
28000	672000	vždy aktuální zdrojové kódy Grsec + podpora
4960	119040	vždy aktuální zdrojové kódy Grsec (60 % sleva pro nekomerční/vzdělávací využití)
11200	268800	vždy aktuální zdrojové kódy Grsec + podpora (60 % sleva pro nekomerční/vzdělávací využití)

Tabulka 6.1: Nabídka Grsec poskytnutá CIV ZČU platná ke dni 17. 5. 2017

7. Srovnání vybraných nadstandardních bezpečnostních řešení

V kapitole 6.2.1 vyjmenované implementace MAC porovnává tabulka 7.1.

	Grsecurity	**AppArmor**	**SELinux**
Identifikace subjektů (a objektů)	cestou	cestou	značkou
Podpora rolí (⇒ RBAC)	✓	✗	✓
Poskytovaná funkcionalita nad rámec AC	NX bit, ASLR, posílení Chroot, ...	✗	✗
Learning mode	✓	✓	✗
Existence grafických nadstaveb nad konfiguračními nástroji	✗	✓	✓
Distribuce, ve kterých je nasazena (ve výchozím nastavení)	Hardened Gentoo	SUSE, Ubuntu [Háj15]	RHEL, Fedora
Využívá (rozhraní definovaná) LSM	✗	✓	✓
Snadno použitelná (pochopitelná)[2]	✓	✓	✗
PaX či jiná forma zvýšené ochrany paměti	✓	✗	✗
Dostupné zdarma	✗[1]	✓	✓

Tabulka 7.1: Srovnání vybraných implementací MAC

7.0.2 Volba bezpečnostního řešení pro podrobnější průzkum

Jedním z úkolů, jehož řešení by měla tato práce přinést, je **návrh na zvýšení úrovně bezpečnosti výpočetního prostředí v rámci CIV za použití komponent analyzovaných řešení**. Jelikož pro otestování možných analyzovaných řešení jsou k dispozici pouze **omezené zdroje**, je nutné provést a zdůvodnit jejich redukci.

Základní posílení bezpečnosti formou doplnění jádra o MAC představují všechny kapitolou 6.2.1 popisované patche, ze kterých však vyčnívá Grsecurity, a to především jím nad rámec MAC poskytovanou funkcionalitou. Grsec slibuje

- zesílení obrany proti převzetí kontroly nad procesem,
- posílení izolace subjektů běžících v chroot prostředí,
- ochranu `/proc` filesystému,
- a další.

Množství modifikací jádra prováděných patchem Grsecurity a zkušenost vedoucího této práce – Michala Švamberga – s jeho používáním představují hlavní důvody volby patche Grsec prostředkem pro plnění zadaných úkolů.

V rámci měření bylo použito jádro verze 4.8.16 a Grsec verze 201701062021 stažená z `https://grsecurity.net/test/grsecurity-3.1-4.8.16-201701062021.patch`.

[1] Parametr se proměnil během psaní tohoto dokumentu. Více viz kapitola 6.6.1.

[2] Tento parametr je samozřejmě subjektivní, nicméně ve svých hodnoceních jej nezapomenou zmínit mnozí uživatelé/recenzenti [Viv09] [AGcomp].

8. Hrozby

Následující pojmy převzaté z [Čer09] jsou pro další pokračování podstatné.

- *Zranitelnost*

  je označení pro zneužitelnou chybu v implementaci čehokoliv. Existence zranitelnosti zapříčiňuje existenci hrozby.
- *Hrozba*

  je označení pro událost zneužívající zranitelnost a narušující tím bezpečnost.
- *Dopad*

  je označuje škodu vzniklou v důsledku působení hrozby.
- *Riziko*

  je pravděpodobnost, že dojde k uplatnění hrozby a vznikne škoda. Hrozba je tím pravděpodobnější, čím snadnější je určitou zranitelnost zneužít.
- *Exploit*

  je sekvence příkazů, které využívají zranitelnost k provedení původně nezamýšlené činnosti typicky za účelem získání prospěchu útočníka.

Jakýkoliv útok vždy směřuje na proces. **Útočník má svůj zájem** přimět proces ke konání toho, co po něm útočník chce čili ovládnout jej čili **spustit kód dle svého přání**. Kapitoly 8.1 až 8.3 klasifikují útoky dle cesty, kterou útočník zvolil k ovládnutí procesu. Každý útok s cílem ovládnutí procesu postavený na manipulaci s AS onoho procesu vyžaduje zásah do toku instrukcí popisovaný v kapitole 8.4.

8.1 Zavedení a vykonání vlastního kódu

Hrozba zavedení (do AS procesu) a vykonání svého – z pohledu procesu tedy cizího – kódu je možná, a to v následujících variantách:

8.1.1 Vytvoření nového spustitelného mapování v AS

Vytvoření nového spustitelného mapování v AS není nic jiného než nahrání mapování ze souboru do paměti a jeho označení spustitelným ať už při spouštění souboru jako takového nebo při dynamickém nahrávání knihoven, se kterými je spustitelný soubor slinkován. Obojí podléhá kontrole přístupových oprávnění AC systémů. Obranou proti útoku tohoto typu je definice přístupových oprávnění na úrovni FS. na jejichž dodržování následně dohlíží AC. Pokud útočník disponuje oprávněním soubor spustit, pak jej spustí a nikdo mu v tom nezabrání.

8.1.2 Modifikace již existujícího zapisovatelného/spustitelného mapování

Útočník

- buďto **modifikuje obsah zapisovatelného a spustitelného mapování** nebo
- **modifikuje obsah pouze zapisovatelného mapování, které spustitelným následně učiní voláním `mprotect()`.**

V každém případě útok předpokládá

- **úspěšné zapsání vlastního kódu do AS procesu** a

  Shellcode je pojmenování kódu vkládaného útočníkem do AS procesu. Ten se zpravidla pokouší o spuštění příkazového interpretru – shellu. Útočník by mohl operace, které chce provést, vpravit již do tohoto kódu, avšak jejich spuštění v samostatném příkazovém interpretru je pro útočníka mnohem snazší. Útočník může svůj shellcode umístit do jakéhokoliv mapování, kde k tomu dostane příležitost, nicméně pouze ve spustitelných mapováních mu bude shellcode platný.
- **ovlivnění toku instrukcí v původním programovém kódu útočníkem**.

Obého útočník dosáhne technikou buffer overflow popsanou v kapitole 8.4. Útoky tohoto typu se nazývají *shellcode injection*. *Shellcode injection* je technika útoku, kdy útočník využije exploit buffer overflow podobně jako u útoku return-to-lib, avšak v tomto případě je **cílem skoku vlastní kód útočníka** [Pel07]. To předpokládá, že jej předtím byl schopen umístit do spustitelného mapování, anebo do mapování, které učinil spustitelným. Obranou proti *shellcode injection* útokům je omezení možnosti spouštění obsahu vybraných mapování, což však může způsobit nefunkčnost

některých aplikací (více viz kapitola 9.4).

8.2 Vykonání existujícího kódu mimo původní programové pořadí

Útoky, při kterých není útočníkem do AS procesu vpravován jeho vlastní strojový kód, avšak útočník zasahuje do toku programu se souhrnně označují jako *return-to-lib*. Útoky tohoto typu vyžadují

- **ovlivnění toku instrukcí v původním programovém kódu** a

  Více viz kapitola 8.4.
- **alespoň částečnou znalost AS napadaného procesu**.

  Útočník má obecně možnost chtít vykonávat obsah umístěný na libovolné adrese. Vzhledem k tomu, že programy používají dynamicky linkované knihovny, jejichž obsah je bez dodatečných bezpečnostních opatření víceméně neměnný a dobře známý, lze přepsáním návratové adresy dosáhnout volání požadované funkce, která ovšem musí v AS být přítomna.

Obranou proti tomuto útoku **nemůže být** technika NOEXEC, neboť útočníkem spouštěný kód je legálně-spustitelným kódem. Obranou je naopak Address Space Layout Randomization, neboť útok tohoto typu vyžaduje znalost podoby adresního prostoru aplikace.

8.3 Vykonání existujícího kódu v původním programovém pořadí s původními daty

Hrozby, které spadají do této kategorie předpokládají jediné – chybu v programovém kódu, čili nedostatečně ošetřený uživatelský vstup.

8.4 Možnosti ovlivnění toku instrukcí v původním programovém kódu

Jak může útočník donutit program k vykonávání jím určeného kódu? **Je-li program napsán správně, útočník ničeho takového obecně nemůže dosáhnout.** Pokud však program obsahuje chybu a umožní útočníkovi zápis vlastní hodnoty do paměti, kterou program následně použije jako adresu příští vykonávané instrukce – adresu skoku, potom pro útočníka existuje možnost, jak ovlivnit tok instrukcí v původním programovém kódu. Zápis vlastní hodnoty do paměti je možný díky technice zvané *buffer overflow*.

Přetečení bufferu, *buffer overflow* či *zápis mimo meze pole* je pojmenování exploitu, který zneužívá zranitelnost „nedostatečně ošetřený zápis do paměti (pole)“. Lépe však tuto zranitelnost pojmenovat jako **„nedostatečně ošetřené vstupy (obecně čehokoliv)“**, neboť jinak by k buffer overflow docházelo při každém vykonávání programu a nejednalo by se tak ani o zranitelnost, jako spíš o vlastnost onoho programu.

Při zápisu mimo meze pole je zapisováno mimo prostor k tomu vyhrazený. Typicky se jedná o zápis za jeho koncem. V jazyce C je tento jev velice snadno dosažitelný, neboť při běhu programu jím definovaném nejsou standardně prováděny žádné kontroly mezí úseků, kam je zapisováno. Existuje-li za pro zápis hodnoty nedostatečně velkým úsekem v paměti prostor procesu přístupný pro zápis, potom zápis pokračuje v něm. Dle mapování, ve kterém k zápisu mimo meze pole došlo se rozlišují

1. **přetečení v userspace-stack**,

   Zápis do lokální proměnné, avšak mimo (čili „i za“) paměťový prostor pro ni vyhrazený vede obecně **k neoprávněné změně** hodnot proměnných, jejichž paměťová místa následují paměťová místa zapisované proměnné. V mapování userspace-stack jimi mohou být

   - návratová adresa dané rozsahové jednotky,
   - adresa předcházejícího aktivačního záznamu,
   - v aktuální rozsahové jednotce lokální proměnné,
   - a případné parametry aktuální funkce.

   Navíc neoprávněný zápis hrozí i všem AZ předcházejícím aktuální AZ, nicméně složitost jejich napadání je výrazně vyšší nežli složitost napadení aktuálního AZ.
2. **přetečení v halda** a

   Změna dynamicky alokované oblasti pro data.
3. **přetečení v BSS či data**.

   Přetečení může vést maximálně k poškození dat.

Buffer overflow je jednou z nejčastěji exploitovaných zranitelností poslední dekády. Netýká se však zdaleka všech programů. Programy napsané v jazycích jako jsou Java, C# či Pascal přetečení na zásobníku detekují a vrhají výjimky. Existují však i jazyky, které kontroly, zda-li zápis náhodnou nesměřuje mimo meze, nedělají – typicky C či C++. Protože se však kontrolami podobného střihu nezabývají, poskytují výkon a jsou proto součástí kódu většiny systémových nástrojů. Tyto nástroje jsou pak obecně náchylnější k výskytu zranitelností zneužitelných útokem buffer overflow. Exploit obvykle sestává z

- **payloadu** a

  Strojový kód vkládaný útočníkem do AS napadaného procesu určený k vykonání.
- **injection vectoru (IV)**.

  Přesměrování toku instrukcí na výše zmíněný payload. Typicky se jedná o modifikaci hodnoty RIP v rámci AZ. Původní odkaz nesený RIP bývá přepsán tak, aby ukazoval zpět do přednějších pozic bufferu, pomocí něhož byla přepsána právě i hodnota RIP, kde útočník má možnost umístit svůj spustitelný kód. V případě nespustitelného zásobníku mu je však jeho kódu v bufferu na zásobníku houby platný.

Běžně používaným vylepšením, vkládá-li útočník svůj vlastní kód a následně na něj i skáče, je tzv. *NOP sliding*. To jest pojmenování postupu, kdy útočník nevyužije bezezbytku prostor daný mu pro jeho shellcode, nýbrž ne nepodstatnou část z něj vyplní prázdnými instrukcemi `NOP`. To útočníkovi usnadní skok do shellcode, neboť mnohdy má útočník možnost adresy pouze odhadovat, nikoliv znát.

8.4.1 Únosy aktivačního záznamu

Při každém volání podprogramu jsou na zásobník ukládány hodnoty RIP a SFP (viz kapitola 4.7). To je z pohledu útočníka nebývalá situace, neboť možnost ovlivnit hodnotu registru EIP a tím i určit příští vykonávanou instrukci po dokončení RJ je zde velmi snadná, samozřejmě „relativně“ snadná.

8.4.1.1 Stack smashing

Útoky zahrnované do kategorie *stack smashing* jsou těmi nejjednoduššími možnými. Prvním zástupcem této kategorie byl Aleph One. Princip je jednoduchý:

1. Na zásobníku je přítomna lokální proměnná, do které je načítán vstup bez kontroly mezí, čili zápisem do této proměnné lze přepsat i paměťová místa za ní následující.
2. Zápisem dojde k přepsání RIP. Útočník typicky hodnotu v RIP – adresu paměti – nasměruje na počátek bufferu, kterým RIP také přepisoval, čímž dosáhne pokusu o vykonávání svých strojových instrukcí, které umístil na zásobník. Samozřejmě za předpokladu spustitelného zásobníku.

8.4.1.2 Off-by-ones & frame pointer overwrites

Off-by-ones je buffer overflow v rámci 1 B. Využívá chyb z nepozornosti, kdy typicky podmínka v cyklu umožní díky chybně napsanému porovnávání (např. `<n` namísto `<=n`) přístup k 1 B paměti. Pochopitelně neoprávněně. Na off-by-ones staví útok pojmenovaný jako *frame pointer overwrite*, což je pojmenování pro útok na první – nejméně významný – bit SFP v rámci aktivačního záznamu. RIP v případě tohoto útoku zůstává netknutý, neboť útočník nemá možnost jej díky SFP, který jej na zásobníku předchází, modifikovat.

8.4.2 Modifikace ukazatele na funkci

Chyba v programu umožňující přepsání hodnoty ukazatele na funkci představuje stejnou zranitelnost jako chyba v programu umožňující přepsání návratové adresy rozsahové jednotky na zásobníku, od které se liší tím, že ukazatele na funkce se mohou vyskytovat ve všech možných mapováních tedy nejen v userspace-stack. Jelikož útočník stěží může modifikovat programový kód, neboť ten nebývá zapisovatelný, vychází jako nejjednodušeji napadení instrukce RET. Instrukce RET totiž nemá žádný operand, neboť adresa, kam bude skončeno je čtena ze zásobníku. Útočníkovi tak stačí změnit ji. Útočník napadá obsah zásobníku. Nejčastěji dochází k napadení návratové adresy rozsahových jednotek.

8.4.3 Spuštění funkcí přítomných v AS procesu

Útočník má k dispozici ke spuštění jakékoliv funkce, které se nacházejí ve spustitelných mapováních v rámci AS procesu. Zejména se tedy jedná o

8.4.3.1 Funkce přítomné v mapování text

Útočník může spouštět funkce definované v rámci kódu vykonávaného programu. Zajímavějším cílem jsou však funkce obsažené ve sdílených knihovnách, neboť ty umožňují přímou manipulaci se zdroji či interakci s jinými procesy systému. Funkce v rámci programu sice také, avšak pouze zprostředkovanou právě funkcemi knihovními.

8.4.3.2 Funkce přítomné v mapování MMS

Mapování MMS může být v rámci AS procesu obecně více. Jedná se o mapování obsahující sdílené knihovny, což znamená, že tyto jsou v hlavní paměti přítomny pouze jednou a procesy je sdílí. Informaci o tom, jaké sdílené knihovny ten který program používá podá program `/usr/bin/ldd` a to **včetně paměťových adres, na kterých jsou tyto knihovny do AS procesu nahrány**.

Kupříkladu takový `/bin/bash` používá

```
ldd /bin/bash
        linux-vdso.so.1 (0x00007ffcf36f8000)
        libncurses.so.5 => /lib/x86_64-linux-gnu/libncurses.so.5 (0x00007fc01389b000)
        libtinfo.so.5 => /lib/x86_64-linux-gnu/libtinfo.so.5 (0x00007fc013671000)
        libdl.so.2 => /lib/x86_64-linux-gnu/libdl.so.2 (0x00007fc01346d000)
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007fc0130c2000)
        /lib64/ld-linux-x86-64.so.2 (0x00007fc013ac0000)
```

- **libdl.so.2**

  Knihovna obsahující rutiny potřebné pro dynamické nahrávání sdílených objektů.
- **libc.so.6**

  Standardní knihovna jazyka C, která poskytuje zapouzdření systémových volání a poskytuje základní knihovní funkce, tak jak je definuje standard jazyka C. Je nezbytná pro běh většiny uživatelských programů. Tato knihovna je velmi oblíbeným cílem útočníků, neboť poskytuje dobře zneužitelné funkce
  - `system()` pro spuštění procesu z cesty,
  - `execve()` pro nahrazení aktuálního procesu jiným či
  - `write()` pro zápis do FS s přístupovými oprávněními procesu, v rámci něhož je volána.
- **ld-linux-x86-64.so.2**

  Dynamický linker nutný pro přilinkovávání dynamických knihoven ke spouštěným programům.

K tomu, aby útočník donutil proces vykonat jím požadovanou funkci, potřebuje znát

- **umístění knihovny v AS procesu** a
- **offset požadované funkce v rámci knihovny**.

  Tuto informaci může útočník získat z tabulky symbolů knihovny ve formátu ELF.

8.5 Převzetí kontroly nad během procesu

Zcizení uživatelské identity předpokládá převzetí kontroly nad během procesu, který disponuje požadovanou identitou, či identitou s oprávněními požadovanou identitu dodatečně získat – superuživatelskou identitou. Útočník převezme kontrolu nad během procesu v okamžiku, kdy jej donutí vykonávat kód, který by bez jeho přičinění proces nevykonával. Toho dosáhne modifikací cílové adresy v libovolné instrukci provádějící skok. Příležitost představují zejména

funkční ukazatele a
: Jsou-li ve VAS procesu přítomny a používány ukazatele na funkce ležící na některé z mapování, vyjmenovaných v kapitole 4.6, která je zapisovatelná.

návratové adresy z rozsahové jednotky (RJ).
: Návratová adresa je součástí každého AZ umístěného na zásobníku. Po ukončení provádění v rámci RJ se skáče do nadřazené RJ.
Přepsání návratové adresy se lze bránit různými způsoby, mezi něž se řadí

uživatelská kontrola mezí polí,

použití výhradně bezpečných funkcí pro práci s řetězci,
: Funkcí, které nespoléhají na přítomnost ukončovacího znaku řetězce – `strncpy()`, ...

použití jazyků bez přímých ukazatelů do paměti či

nasazení kanárka.
: *Kanárek* je součástí mechanismu umožňujícího detekci přepsání návratové adresy (NA) aktivačního záznamu (AZ). Vykonávání kódu rozsahové jednotky (RJ) sestává z

prologu,
: Mimo vytváření AZ se v této fázi umisťuje do AZ hodnota kanárka.

vykonávání kódu rozsahové jednotky a
V případě vyvinutí snahy o přepsání NA je přepsán i kanárek ležící těsně před ní.

epilogu.
Má-li kanárek hodnotu odlišnou od té, kterou byl inicializován, je detekován pokus o přepsání NA.

Problémem tohoto mechanismu je fakt, že získá-li útočník hodnotu kanárka, kterou byl inicializován, po změně NA jej může na danou hodnotu opět nastavit. Implementace mechanismu kanárka se tomuto konání snaží různými způsoby předejít, zatímco útočník se snaží všemožné ochrany hodnoty kanárka prolamovat. V konečném důsledku není na kanárky stoprocentní spolehnutí [HI12].

Modifikaci návratové adresy **lze považovat za převzetí kontroly** nad během procesu,nicméně skok na náhodnou adresu běh procesu s největší pravděpodobností akorát rychle ukončí, neboť na adrese ve VAS procesu může **vyjma instrukce provádějící volání** útočníkem požadované funkce ležet nic, data či náhodná (⇒ útočníkovi nehodící se) instrukce. Takový přístup do VAS v prvním případě vyvolá výjimku Page fault, kterou obslouží jádro zasláním signálu procesu, který se pokoušel číst z nedostupných paměťových míst. Ve zbylých dvou případech běh procesu také nebude mít dlouhé trvání a skončí patrně na výjimce „neplatná instrukce“ či výjimkou Page fault. Útočník musí adresu příští vykonávané instrukce volit s rozmyslem. Obecně má dvě možnosti:

- skok na vlastní kód nebo
- skok na v programu již existující funkci.

8.5.1 Skok na vlastní kód

Útočník může iniciovat vykonávání strojového kódu, jenž do VAS dostal svým vlastním přičiněním. Strojový kód může umístit do jakéhokoliv segmentu, do kterého je povolen zápis. Typicky bývá umisťován do staticky alokovaného prostoru pro proměnné na zásobníku či do dynamicky alokovaného prostoru na haldě. Aby byl vložený kód k něčemu platný, musí pochopitelně respektovat instrukční sadu srozumitelnou procesoru, na němž je vykonáván napadený proces. Strojový kód vložený útočníkem do adresního prostoru procesu je označován jako „shellcode“.

8.5.2 Skok na v programu již existující funkci

Útočník též může skokem iniciovat vykonávání libovolné ve VAS již přítomné funkce. K tomuto musí naprosto přesně znát její adresu. Uspořádání VAS procesu však při nenasazení techniky *Address Space Layout Randomization* (*ASLR*) (viz kapitola 9.1) je pokaždé stejné, navíc zjistitelné přečtením souboru `/proc/PID/maps`.

Útočník zpravidla vyhledává funkce, jejichž volání pro něj má nejvyšší „přidanou hodnotu“. Mezi takové patří například

`system()` či
Funkce `system()` z knihovny `libc` slouží ke spuštění příkazu zadaného jako její parametr v implicitním shellu uživatele. Program, z něhož byla funkce `system()` volána je uspán a znovu se začne vykonávat až po návratu z ní.

`execve()`.
Funkce `execve()` z knihovny `libc` nahradí aktuálně prováděný proces jiným dle jí předaných parametrů.

8.6 Únik informace z kernelspace-stack

Ke každému vláknu v User Space existuje v Kernel Space právě jeden kernelspace-stack do nějž jsou ukládány záznamy o systémových voláních týkajících se vykonávání daného vlákna. Jak se stane, že se z kernelspace-stack čili Kernel Space dostanou data do User Space?

1. Předpoklad: Kernelspace-stack obsahuje data, která v něm z zůstala z poslední obsluhy systémového volání. Sice formálně nedostupná, avšak fyzicky nesmazaná.
2. Útočník vynutí opětovné systémové volání téhož vlákna. Do kernelspace-stack se tak dostane záznam odpovídající novému systémovému volání nesoucí jemu typickou datovou strukturu.
3. Není-li daná datová struktura řádně inicializována, může se stát že některá její komponenta či struktura **převezme data, která na zásobníku leží ještě z dob obsluhy předchozího systémového volání**, neb ta nebyla fakticky smazána.
4. Pokud jádro strukturu kopíruje do User Space jako součást svého výstupu, potom se útočník dostal k požadovaným datům.

Avšak data zbylá po obsluhách předchozích systémových volání nic právě zneužitelného zpravidla nenabízejí. Navíc jeho obsah není snadné predikovat. Co však kernelspace-stack obsahuje vždy jsou ukazatele, kterými si „udržuje svou strukturu“. Z jakéhokoliv takového ukazatele, který se podaří útočníkovi získat si útočník může dopočítat adre-

su počátku zásobníku:

$$\begin{aligned}
\texttt{kstack_base} &= \texttt{address} \;\;\&\;\; \sim (\texttt{THREAD_SIZE} - 1) \\
\texttt{kstack_base} &= \texttt{address} \;\;\&\;\; \sim (8192_{10}{}^{1} - 1) \\
\texttt{kstack_base} &= \texttt{address} \;\;\&\;\; \sim (00000000.00000000.00100000.00000000_2 - 1) \\
\texttt{kstack_base} &= \texttt{address} \;\;\&\;\; \sim 00000000.00000000.00011111.11111111_2 \\
\texttt{kstack_base} &= \texttt{address} \;\;\&\;\; 11111111.11111111.11100000.00000000_2
\end{aligned}$$

Po získání adresy počátku kernelspace-stack pro dané vlákno má útočník adresu struktury `current_thread_info`, neboť ta je umístěna na samém dně zásobníku. Další možný postup útočníka využívá znalostí získaných z této datové struktury [OberRos].

8.7 Neoprávněné opuštění chroot prostředí

Existuje řada způsobů, jak opustit příkazem chroot prostředí:

8.7.1 Vytvořením vnořeného chroot prostředí

1. Uživatel v chroot prostředí získá identitu privilegovaného uživatele.
2. Uživatel vytvoří podadresář, na který zavolá `chroot()`.
3. Dojde ke změně chroot prostředí
4. Adresář nadřazený podadresáři na němž došlo k vytvoření nového chroot prostředí prostředí se již nachází mimo chroot prostředí.
5. Uživatel se může pohybovat po celém souborovém systému.

8.7.1.1 Připojením zařízení

Získá-li subjekt uzavřený v chroot prostředí identitu, která mu zajistí přístupové oprávnění číst soubor zařízení, může toto oprávnění využít k připojení zařízení, respektive k přístupu k objektům ležícím mimo chroot prostředí.

8.8 ROP

Return-Oriented Programming (ROP) je technika zřetězení tzv. ROP gadgetů do tzv. ROP chainu. Rozděluje shellcode na části a byla vynalezena za účelem převzetí kontroly nad procesem fungující i v případě aktivní ochrany proti vykonávání obsahu oblasti userspace-stack ve VAS daného procesu [Zah11]. Cílem exploitů postavených na technice ROP bývá volání funkce, která obsah stacku učiní opět spustitelným – `mprotect()`.

Podrobnější vysvětlení postupu při útoku ROP technikou lze nalézt v prezentaci pánů Buchanana, Roemera, Savageho a Shachama – Return-oriented Programming: Exploitation without Code Injection [Buc+08].

8.8.1 ROP chain

ROP chain je **posloupnost** [Zah11] sestávající z

- adres počátků *ROP gadgetů* a
- výplní sloužících pro `pop` instrukce v epilog.

Adresy počátků ROP gadgetů jsou ve vhodném pořadí umístěny na zásobník použitím instrukcí `push`.

8.8.2 ROP gadget

ROP gadget je **posloupnost instrukcí zakončená instrukcí** `ret` nacházející se ve spustitelné oblasti VAS využívaná technikou ROP k sestavení kódu exploitu. ROP gadget sestává z

epilog funkce a

Podoba epilogu funkce na architektuře x86 je zachycena strojovým kódem x86 8.1, kde

- `%rbp` = ukazatel na začátek AZ ležící na vrcholu zásobníku,
- `%rsp` = ukazatel na vrchol zásobníku (konec téhož AZ),
- `movq %A %B` = přesun „quadword“ – 64 bitů – z registru `A` do registru `B`,
- `popq %rbp` = načtení „quadword“ hodnoty ze zásobníku a zvýšení jeho vrcholu – `%rsp` a
- `ret` = odebrání adresy ze zásobníku a skok na ni.

jedné instrukce před začátkem epilog.

Touto instrukcí může být libovolná instrukce včetně `call` knihovní funkce.

[1] Velikost zásobníku vyhrazeného pro jedno vlákno bývá zpravidla 8 kB (či ojediněle 4 kB) [OberRos].

```
movq %rbp, %rsp
popq %rbp
ret
```

Strojový kód x86 8.1: Epilog funkce na architektuře x86

8.9 Zabezpečení Kernel Space

Útoků zaměřených výhradně na programy běžící v user mode ubývá [KPK14] a útočníci se čím dál tím více pokoušejí pro dosažení svých cílů využít, čili exploitovat, zranitelnosti v Kernel Space neboli zranitelnosti jádra samotného [OberRos]. Níže jsou zmíněny možné příčiny toho, proč dnes již útoky na programy běžící v User Space nejsou tak atraktivní jako byly dříve:

1. Exploitování procesů běžících v user mode je v důsledku nasazení dnes již mnoha všemožných ochran příliš náročné (v porovnání s exploitováním jádra samotného). Některé z ochran nasaditelných na user mode programech tento dokument zmiňuje:
   - ASLR – znáhodňování některých mapování (viz kapitola 9.1),
   - použití kanárků (viz kapitola 8.5),
   - NOEXEC – nespustitelnost vybraných mapování (viz kapitola 9.2.1.1),
   - chroot – izolace procesu do chroot prostředí (viz kapitola 5.4)
   - a další ...

   A mnohé ponechává bez povšimnutí:
   - *RELocation Read-Only (RELRO)* – relokace vybraných mapování ELF souboru [Obe11] nikoliv nepodobná RANDMMAP.
   - *FORTIFY_SOURCE* – detekce přetečení zásobníku v C/C++
   - a mnoho dalších ...

   Otázkou je, proč je zabezpečení Kernel Space pozadu za zabezpečením User Space? Odpověď viz body 2, 3 a 6.
2. Cílem útočníka je získání superuživatelských oprávnění pro konkrétní proces běžící v user mode. Zpočátku mu k tomu stačilo exploitovat uživatelské procesy samotné. Exploitování jádra bylo zřejmě ve většině případů náročnější a úsilí do něj vložené se tak nevyplácelo a proto se nekladl důraz ani na jeho zabezpečení.
3. Jádro je dnes se svými 18,3 miliony řádků kódu [counter] natolik komplexním softwarem, že o zranitelnosti v něm není nouze[2].
4. Získání kontroly nad jádrem samotným představuje pro útočníka vyšší potenciální zisk, neboť právě v Kernel Space jsou implementovány bezpečnostní mechanismy a je tedy možná i jejich centrální deaktivace.
5. Jádro již z principu nemůže být tak dokonale chráněno jako programy běžící v user mode, neb případné ochrany kernel mode mohou být umístěny pouze v Kernel Space a jejich obejití/deaktivace nemá možnost být tak dobře střežena – neexistuje žádná vyšší moc, která by hlídala dodržování přístupových oprávnění v jádře.
6. Každé rozhraní interagující s „vnějším prostředím“ poskytuje potenciální prostor pro vznik chyby a tedy i zranitelnosti zneužitelné útočníkem. Jádro poskytuje takových rozhraní nepřeberné množství[3].
7. Odstranění zranitelnosti z jádra vyžaduje jeho výměnu, nejedná-li se o zranitelnost existující v rámci některého z jeho modulů. Výměna jádra je náročnější operací než výměna uživatelského programu a proto existuje vyšší pravděpodobnost, že zranitelnost bude déle k dispozici.

Výše zmíněné body představují nejpravděpodobnější příčiny většího počtu objevených zranitelností [CVE] a s ním spojeného většího počtu pokusů o jejich exploitování v posledních letech.

8.9.1 Jádro není cílem

Převzetí kontroly nad samotným jádrem je jistě pro mnohé zábavné, nicméně typickým cílem útočníka kočírování jádra není. Útočník chce jediné: získat pro svůj proces běžící v user mode superuživatelská oprávnění. Převzetí kontroly nad jádrem je pouze prostředkem k tomu, jak toho útočník dosáhne. Útok na jádro má svá specifika. Útočník si musí dát obzvláště pozor na skutečnost, že jakékoliv jeho pochybení bude potrestáno pádem celého systému. Postup útočníka lze shrnout do těchto bodů:

1. Nalezení zranitelnosti v jádře.
2. Manipulace se vstupy jádra vedoucí k využití této zranitelnosti čili k **vykonání kódu dle volby útočníka**.
3. Změna úrovně oprávnění procesu dle volby útočníka.
4. Navození „řádného návratu“ do user mode čili do procesu, který manipuloval vstupy jádra v bodě 2.
5. Profit z běhu procesu, jemuž byla změněna úroveň oprávnění v bodě 3.

[2] Odhalených zranitelností v linuxu objevených v posledních letech stále přibývá [CVE]. Zvýšený počet odhalených zranitelností s sebou nese i zvýšený zájem tyto zranitelnosti exploitovat.

[3] Rozhraní, kterými komunikuje jádro se zbytkem systému, je spousta. Například systémová volání, ovladače zařízení, pseudo-filesystémy, atd.

8.9.2 Útok ret2usr

Útočník umisťuje kód, který chce, aby jádro vykonalo do User Space, neboť to je pro útočníka nejjednodušší možný způsob. Paměť uživatelského prostoru totiž zůstává namapovaná, i když procesor běží v kernel mode. Útočníkovi tedy stačí „pouze“ přesvědčit jádro, aby skočilo na adresu ležící v uživatelském prostoru [PKK14].

Return to userspace (ret2usr) útok je postavený na principu **přepsání ukazatele v rámci Kernel Space tak, aby směřoval do User Space**. Není podstatné zda-li modifikovaný ukazatel je ukazatelem na funkci či na data. V obou případech jde o ret2usr útok. Odkazovanou oblast v User Space má útočník možnost přizpůsobit konkrétním svým potřebám. Použije-li se obsah, na který ukazuje podvržený ukazatel z Kernel Space, jako kód funkce, potom je pochopitelně vykonáván s veškerými poctami náležícími kódu běžícímu v Ring 0 – **nevztahují se na něj žádné kontroly** a může tedy provádět cokoli [KPK14]. Ochrana programů běžících v user mode před jádrem pochopitelně neexistuje, neb v systému není nikdo, kdo by ji byl schopen vymoci.

8.9.2.1 Útok ret2dir

Existuje celá řada ochran, která efektivně brání provedení ret2usr útoku – viz kapitola 9.5. Existuje proto i varianta útoku ret2usr, proti které jsou ochrany (vyjma ochrany XPFO) neúčinné [PKK14]. Jmenuje se *Return to direct mapped memory (ret2dir)*.

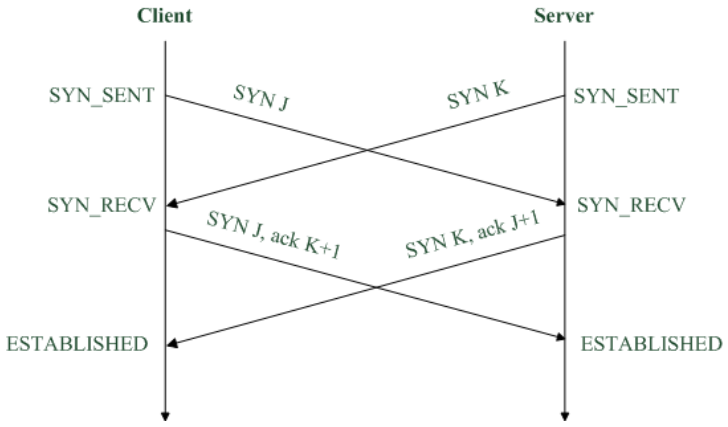
Paměť User Space je dostupná jádru prostřednictvím tabulky stránek. Bezpečnostní řešení vyjmenovaná v kapitole 9.5 blokují přístup do User Space právě na úrovni tabulky stránek [PKK14]. Útok ret2dir tyto ochrany obchází tím, že k přístupu do paměti nepoužívá tabulku stránek. Využívá faktu, že x86-64-kompatibilní jádra udržují lineární mapování celého rozsahu fyzické paměti – tzv. „přímé mapování“ v rámci Kernel Space. Dělají tak z různých důvodů včetně z důvodu potřeby správy paměti na úrovni stránek. Každá fyzická stránka přítomná v systému má vlastní adresu mapovanou v přímém mapování do Kernel Space, kde je její obsah vykonatelný jádrem. Přinutí-li útočník jádro skočit na přímo mapovanou adresu, mechanismy zabraňující přístupu do User Space toto ponechají bez povšimnutí, přestože cílová adresa odpovídá stránce v uživatelském prostoru. **Existence přímého mapování představuje zranitelnost, která umožňuje obejití/nezafungování ochran proti útokům typu Return to userspace.**[4]

Útočník však musí vyvinout zvýšené úsilí oproti běžnému útoku ret2usr, neboť musí být schopen určit adresy fyzických stránek, které obsahují kód, jehož vykonání si přeje. Informace o fyzických adresách stránek procesu s daným PID lze[6] zjistit ze souborového systému `/proc` v souboru `/proc/<PID>/pagemap`.

8.10 Souběžné spojení

Souběžné spojení je TCP protokolem uznávaný způsob navázání spojení definovaný přímo v RFC 793. Znázorněný je na obrázku 8.1. Souběžné spojení nastane, když

- TCP spojení je ve stavu `SYN_SENT`
  ⇒ Čeká se na `SYN/ACK` paket od protistrany.
- a zároveň od protistrany přijde, namísto očekávaného `SYN/ACK` paketu, paket `SYN`.



Obrázek 8.1: Navázání spojení prostřednictvím TCP Simultaneous Connect

[5] Možnost přistupovat na libovolné místo v hlavní paměti skončila v jádře verze 3.9. Řešením pro útočníka na novějších jádrech je technika zvaná ROP popsaná v kapitole 8.8.

[7] Tyto soubory lze pro běžné uživatele znepřístupnit, avšak pravidlem to není.

8.10.1 DoS útok na službu postavený na existenci TCP Simultaneous Connect

Možnosti souběžného navázání spojení lze využít k znepřístupnění služby běžící na několika málo veřejně známých adresách. Útok vypadá následovně:

1. Útočník zná adresu, na které je poskytována služba.
2. Útočník detekuje `SYN` paket klienta směřující na adresu služby.
3. Útočník klientovi následně pošle `SYN` paket vydávajíc se za server poskytující službu.
4. Klientovi útočníkův paket přijde ve chvíli, kdy je spojení na straně klienta ve stavu `SYN_SENT`.
   ⇒ Klient akceptuje sekvenční číslo „serveru“ bez jeho ověření.
5. Skutečné sekvenční číslo serveru klient následně již neakceptuje.

9. Grsecurity/PaX

Grsecurity (Grsec) je soubor modifikací jádra posilujících jeho bezpečnost distribuovaný formou patche. Jak v popisu, tak i v následném měření je uvažováno Grsec ve verzi 201701062021 aplikovatelné na jádra ve verzích 3.1 - 4.8.16. Kořenem veškerých nastavení dodaných patchem Grsecurity jádru je větev `/Security options/Grsecurity`. Modifikací jádra obsažených v Grsec je velmi mnoho a není zde prostor je všechny pojmout. Proto byly vybrány pouze ty „zajímavější“ z hlediska nasazení na CIV:

- PaX – soubor preventivně působících bezpečnostních technik týkajících se hlavní paměti (viz kapitoly 9.1 až 9.16),
- zvýšení zabezpečení chroot prostředí,
- přidání podpory MAC/RBAC,
- a několik dalších.

Page eXec project (PaX) je soubor technik, které aktivně brání možnému zneužití zranitelností v podobě softwarových chyb umožňujících útočníkovi získat neomezený přístup do adresního prostoru napadeného procesu. PaX obecně nezjišťuje zranitelnosti, pouze preventivně omezuje techniky, kterými lze těchto zranitelností zneužít. PaX sestává bezpečnostních technik rozdělitelných zhruba do čtyřech hlavních oblastí:

vylepšuje řízení přístupu (AC) k obsahu VAS,
: Na architekturu x86-32 přináší PaX možnost zamezit vykonávání obsahu stránek emulací NX bitu. Na novějších architekturách, kde je NX již přítomen, jej PaX využívá. Jím posílené AC ve výchozím nastavení odebírá oprávnění zapisovat do stránek obsahujících instrukce a oprávnění vykonávat obsah stránek obsahujících data.

činí obtížnějším útok převzetí kontroly,
: Znáhodňováním adres počátků jednotlivých mapování (ASLR) v rámci VAS PaX znepříjemňuje doskok útočníka, který tak již nevystačí s pouhou znalostí umístění funkce ve VAS.

věnuje se i zabezpečení přímo v Kernel Space a
: Preventivně maže kernelspace-stack, aby předešel datovým únikům z něj. Dále PaX do jisté míry znemožňuje provádění kódu umístěného v User Space v kernel mode. Dosahuje toho důsledným oddělením User Space a Kernel Space. Úspěchu útoků ret2dir však předejít nedokáže.

dokáže detekovat pokusy o neoprávněnou manipulaci s hlavní pamětí.
: Iniciátora takové manipulace s hlavní pamětí ukončuje.

9.1 ASLR

ASLR (Address Space Layout Randomization) je technika znáhodňující [Ass] rozmístění jednotlivých mapování po AS procesu a tím **znesnadňující** zneužití zranitelnosti přetečení zásobníku. ASLR neodstraňuje příčinu zranitelnosti – chybný kód jednotlivých programů, nýbrž působí jako prevence, která znesnadní zneužití této zranitelnosti. Proto je ASLR potencionálně efektivní ochranou i proti doposud neznámým hrozbám. ASLR je podstatné v ochraně proti útokům vyžadujícím znalost adres v AS procesu jako např. adresy různých knihoven.

Útoky vedoucí k ovládnutí procesu vyžadují zpravidla předchozí znalost AS procesu. Počátek každého mapování je v rámci AS procesu proto volen náhodně při každém novém spuštění procesu, což zásadně sníží šance útočníka na úspěšné převzetí kontroly nad procesem, neboť skok na adresu, kde se nenalézá předpokládaná instrukce, čili skok na adresu bez smysluplného obsahu povede spíše ke zhroucení procesu, nežli k převzetí kontroly nad procesem. Zhroucení procesu lze následně snadno detekovat a na něj reagovat. Náhodná volba adres jednotlivých mapování v AS procesu vede u mnoha pokusů o převzetí kontroly nad procesem k selhání procesu.

Předpovídatelnost rozložení paměti využívaná hrozbami 8.2 je při použití PaX-ASLR redukována náhodným umísťováním počátků všech možných mapování v rámci AS **pro každou jednu instanci programu**.

Náhodné umisťování mapování po AS procesu by za stavu, kdy pro útočníka není složité se dozvědět o AS jím napadaného procesu podrobnosti ze souborového systému `/proc`, nebylo až takovým přínosem. Proto Grsecurity posiluje bezpečnost i v tomto směru – viz kapitola 9.21.

9.1.1 ASLR není exkluzivní pro PaX

Technika ASLR existuje nezávisle na PaXu, nicméně v méně pokročilých formách [Mül08] (viz tabulka 9.1).

Nosič	První zaznamenaný výskyt	Komentář
Linux	verze 2.6.12 (cca červen 2005)	Znáhodňují se pozice pouze anonymních mapování – zásobníku a haldy.
jádro OS Windows	verze 6.0 (cca červen 2006)	Pouze pro explicitně označené spustitelné soubory.
PaX	2000	Znáhodnění počátku mapování je možné pro každé mapování. Vše se odvíjí od požadavků specifikovaných v Executable and Linking Format (ELF) hlavičkách spustitelných souborů, jejichž obsah bude vykonáván procesy, jejichž AS bude znáhodňován. PaX podíl „náhody“ na umístění jednotlivých mapování výrazně umocňuje.

Tabulka 9.1: Výskyt ASLR

9.1.2 PaX-ASLR funguje pouze u vybraných procesů

PaX-ASLR se aplikuje výhradně na procesy, které

- **vykonávají spustitelné soubory ve formátu ELF** a
- **používají (dynamicky linkují) výhradně knihovny ve formátu ELF**.

PaX dále předpokládá určitý formát spustitelného souboru, jež je procesem vykonáván, v závislosti na čemž následně randomizace funguje či nikoliv.

Position Independent Executable (PIE) je spustitelný soubor obsahující pozičně nezávislý kód (PIC).

Position Independent Code (PIC) je pozičně nezávislý kód je strojový kód, který je možné vykonat nezávisle na tom, na jaké adrese je v operační paměti umístěn.

9.1.2.1 Příklad

Příklad v rozdílném přístupu PaXu je zachycen C kódem 9.1 kompilovaným příkazy 9.1 a spuštěným v obou variantách na systému s PaX a bez něj s výsledkem uvedeným v tabulce 9.2.

```c
#include <stdio.h>

void doit() { return 0 ; }

int main() {
        printf("main @ %p\n", main);
        printf("doit @ %p\n", doit);
        return 0;
}
```

C kód 9.1: Program pro demonstraci rozdílného mapování na systémech s/bez PaX

```
gcc -o aslr-test-withpie -pie aslr-test.c
gcc -o aslr-test-without aslr-test.c
```

Příkaz/y 9.1: Kompilace programu do formátu PIE a mimo něj

9.1.3 Příklad uspořádání AS procesu

Obsah AS lze vyčíst z pseudo-souborového systému `/proc`, konkrétně obsah vlastního AS je procesu dostupný v souboru `/proc/self/maps`. Příklad podoby AS pro konkrétní proces (vykonávající soubor `/bin/dd`) je uveden v příkazu 9.2. Popis jednotlivých oblastí je uveden v tabulce 9.3.

9.1.4 ASLR jako obrana proti útoku

Jednotlivá mapování jsou na sobě nezávislá. Díky tomu lze v rámci ASLR posouvat jednotlivá mapování nezávisle na sobě a dále tak snižovat pravděpodobnost úspěchu útočníka, neboť každá oblast může být posunuta o jinou „vzdálenost“ a útočník je nucen hádat více adres. Útočník se může dozvědět podrobnosti o AS procesu

Přítomnost PaX v systému	Program ve formátu PIE	Výstup programu
ne	ne	vždy stejný
ne	ano	vždy stejný
ano	ne	vždy stejný
ano	ano	vždy nestejný

Tabulka 9.2: Výstup programu zachyceném v C kódu 9.1 při opakovaném spuštění

```
$ $ /bin/dd if=/proc/self/maps of=/dev/stdout 2>/dev/null
A 00400000-0040e000 r-xp 00000000 08:11 788032                           /bin/dd
B 0060d000-0060e000 r--p 0000d000 08:11 788032                           /bin/dd
C 0060e000-0060f000 rw-p 0000e000 08:11 788032                           /bin/dd
D 01c34000-01c55000 rw-p 00000000 00:00 0                                [heap]
E 7efc11b15000-7efc11cb6000 r-xp 00000000 08:11 2626963
 ↪  /lib/x86_64-linux-gnu/libc-2.19.so
F 7efc11cb6000-7efc11eb6000 ---p 001a1000 08:11 2626963
 ↪  /lib/x86_64-linux-gnu/libc-2.19.so
G 7efc11eb6000-7efc11eba000 r--p 001a1000 08:11 2626963
 ↪  /lib/x86_64-linux-gnu/libc-2.19.so
H 7efc11eba000-7efc11ebc000 rw-p 001a5000 08:11 2626963
 ↪  /lib/x86_64-linux-gnu/libc-2.19.so
I 7efc11ebc000-7efc11ec0000 rw-p 00000000 00:00 0
J 7efc11ec0000-7efc11ee0000 r-xp 00000000 08:11 2626959
 ↪  /lib/x86_64-linux-gnu/ld-2.19.so
K 7efc12046000-7efc120bb000 r--p 00000000 08:11 1051449
 ↪  /usr/lib/locale/locale-archive
L 7efc120bb000-7efc120be000 rw-p 00000000 00:00 0
M 7efc120de000-7efc120e0000 rw-p 00000000 00:00 0
N 7efc120e0000-7efc120e1000 r--p 00020000 08:11 2626959
 ↪  /lib/x86_64-linux-gnu/ld-2.19.so
O 7efc120e1000-7efc120e2000 rw-p 00021000 08:11 2626959
 ↪  /lib/x86_64-linux-gnu/ld-2.19.so
P 7efc120e2000-7efc120e3000 rw-p 00000000 00:00 0
Q 7ffdc0f6c000-7ffdc0f8d000 rw-p 00000000 00:00 0                        [stack]
R 7ffdc0f8f000-7ffdc0f91000 r-xp 00000000 00:00 0                        [vdso]
S 7ffdc0f91000-7ffdc0f93000 r--p 00000000 00:00 0                        [vvar]
T ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                [vsyscall]
```

Příkaz/y 9.2: Výpis oblastí přítomných v AS procesu `/bin/dd`. Velká písmena na počátku řádků jsou do výstupu příkazu `/bin/dd` doplněna za účelem snadnějšího odkazování se na ně.

Oblast	Význam
A	Čitelná a spustitelná oblast původem z `/bin/dd`. ⇒ Jedná se o mapování text obsahující strojový kód programu.
B	Čitelná oblast původem z `/bin/dd`. ⇒ Jedná se o mapování ??? obsahující konstanty.
C	Čitelná a zapisovatelná oblast původem z `/bin/dd`. ⇒ Data.
D	Halda (řečeno rovnou).
E	Mapování s právy RX původem z `libc-2.19.so`. ⇒ Kód knihovny `libc`.
F	Mapování bez práv původem z `libc-2.19.so`.
G	Mapování s právem R původem z `libc-2.19.so`. ⇒ Inicializovaná data knihovny `libc`.
H-I	Mapování s právy RW původem z `libc-2.19.so`. ⇒ Neinicializovaná data knihovny `libc`.
J	Mapování pro čtení a spouštění původem z `ld-2.19.so`. ⇒ Jedná se o kód dynamického linkeru.
K-M	Soubor `/usr/lib/locale/locale-archive` obsahuje národní verze řětezců. Jedná se o mapování určené pouze pro čtení.
N	Mapování pouze pro čtení původem z `ld-2.19.so`. ⇒ Jedná se o inicializovaná data dynamického linkeru.
O-P	Mapování pro čtení a zápis původem z `ld-2.19.so`. ⇒ Jedná se o neinicializovaná data dynamického linkeru.
Q	Zásobník (řečeno rovnou).

Tabulka 9.3: Oblasti mapované do AS procesu vykonávající soubor `/bin/dd`

- **díky chybě v programu**,

  Z hlediska útočníka nejlepší stav. Proces na základě vnějších či dokonce i žádných stimulů útočníkovi vyzradí potřebné informace o svém AS. Útočníkovi v takovém případě pochopitelně ASLR nezabrání v jeho úmyslech.

- **hrubou silou** nebo

  Útok hrubou silou je útok, při kterém je útočníkovi umožněno využití jeho znalostí nabytých v předchozích útocích na tentýž objekt zájmu čili útočník se může „učit“. Hrubou silou má smysl útočit v případě stále stejné randomizace AS procesu, např. na síťové démony, které při každém novém spojení zavolají pouze `fork()`, neboť `fork()` zkopíruje AS rodiče do AS potomka čili zachová randomizaci uspořádání z rodiče. Naproti tomu `execve()` pochopitelně randomizaci nahradí randomizací novou.

- **hádáním**.

  Útok hádáním je ideální stavem (z pohledu obránce). K hádání se útočník uchýlí v případě, že není v jeho silách se „učit“. Jelikož uspořádání mapování v AS procesu je pokaždé jiné, útočník bude mít při každém útoku stejnou (nepatrnou) pravděpodobnost úspěchu.

9.1.4.1 Vyčíslení pravděpodobnosti úspěchu útoku

Nechť

R_s je počet náhodně volených bitů v adrese počátku mapování zásobníku,
R_m je počet náhodně volených bitů v adrese počátku mapování vzniklého voláním funkce `mmap()`,
R_x je počet náhodně volených bitů v adrese počátku mapování kódu,
L_s je nejméně významná pozice z pozic náhodně volených bitů mapování zásobníku,
L_m je nejméně významná pozice z pozic náhodně volených bitů mapování vzniklého voláním funkce `mmap()`,
L_x je nejméně významná pozice z pozic náhodně volených bitů mapování kódu,
A_s je počet náhodně volených bitů mapování zásobníku napadených v jednom pokusu,
A_m je počet náhodně volených bitů mapování vzniklého voláním funkce `mmap()` napadených v jednom pokusu,
A_x je počet náhodně volených bitů mapování kódu napadených v jednom pokusu,
α je počet provedených pokusů,
N je celkový počet náhodně volených bitů, které musí útočník uhádnout,
$P_g(\alpha)$ je pravděpodobnost úspěchu útoku hádáním při α pokusech a
$P_b(\alpha)$ je pravděpodobnost úspěchu útoku hrubou silou při α pokusech.

Architektura	R_s	R_m	R_x	L_s	L_m	L_x
IA-32	24	16	16	4	12	12

Tabulka 9.4: Parametry ASLR pro různé architektury

V případě architektury IA-32 se v adrese počátku zásobníku (vzhledem k hodnotám uvedeným v tabulce 9.4) náhodně volí podtržené bity z následující posloupnosti bitů:

$$0, 1, 2, 3, \underline{4}, \underline{5}, \underline{6}, \underline{7}, \underline{8}, \underline{9}, \underline{10}, \underline{11}, \underline{12}, \underline{13}, \underline{14}, \underline{15}, \underline{16}, \underline{17}, \underline{18}, \underline{19}, \underline{20}, \underline{21}, \underline{22}, \underline{23}, \underline{24}, \underline{25}, \underline{26}, \underline{27}, 28, 29, 30, 31$$

Pro celkové množství náhodně volených bitů (entropii), které musí útočník uhádnout – N – platí:

$$N = (R_s - A_s) + (R_m - A_m) + (R_x - A_x)$$

9.1.4.2 Vyčíslední pravděpodobnosti úspěchu útoku při hádání

Nechť

β	je pravděpodobnost jevu uhádnutí jednoho bitu (z adresy),
γ	je pravděpodobnost jevu uhádnutí n bitů (z adresy) zároveň,
δ	je pravděpodobnost opačného jevu k předchozímu jevu čili neuhádnutí n bitů (z adresy).

Bit je binární veličina, z čehož plyne že jeho uhádnutí je pravděpodobné z 50 procent:

$$\beta = \frac{1}{2}$$

Uhádnutí n binárních čísel zároveň představuje uhádnutí každého jednoho z nich. To znamená, že pravděpodobnost γ bude sjednocením n pravděpodobností β:

$$\gamma = \left(\frac{1}{2}\right)^n$$

Veličinu γ lze zapsat také následovně:

$$\gamma = 2^{-n}$$

Má-li útočník na hádání α pokusů, potom je vhodné naformulovat δ:

$$\delta = 1 - 2^{-n}$$

Pravděpodobnost uhádnutí je 1 bez pravděpodobnosti neuhádnutí během α pokusů:

$$P_g(\alpha) = 1 - \delta^{\alpha}$$

Výsledný vzorec P_g je patrný z rovnosti 9.1. Pro různé hodnoty N a α vyčísluje P_g tabulka 9.5. Případné hodnoty 0 a 1 v ní uvedené jsou pouze aproximacemi jim velmi blízkých hodnot. Tabulka byla převzata z [03].

$$P_g(\alpha) = 1 - (1 - 2^{-N})^{\alpha}$$

Rovnost 9.1: Pravděpodobnost úspěchu útoku hádáním P_g při hádání N bitů adresy a α násobném opakování

9.1.4.3 Vyčíslení pravděpodobnosti úspěchu útoku hrubou silou

Při útoku hrubou silou lze vyjít z γ čili pravděpodobnosti jevu uhádnutí n bitů (z adresy) zároveň odvozeném výše:

$$\gamma = 2^{-n}$$

Útok hrubou silou předpokládá, že každý jeden útok závisí na útoku jemu předcházejícím, neboť každý případný neúspěch zmenšuje množinu možností pokusu dalšího čili pokusy mají paměť. Výsledný vzorec P_b je patrný z rovnosti 9.2.

Pro jednotlivá mapování existují různé modifikace ASLR:

hádaných bitů (N)	počet opakování útoku (α)													
	1	4	16	64	256	2^{10}	2^{14}	2^{18}	2^{20}	2^{24}	2^{32}	2^{40}	2^{56}	2^{64}
1	0,50	0,94	1	1	1	1	1	1	1	1	1	1	1	1
2	0,25	0,68	0,99	1	1	1	1	1	1	1	1	1	1	1
4	0,06	0,23	0,64	0,98	1	1	1	1	1	1	1	1	1	1
8	0	0,02	0,06	0,22	0,63	0,98	1	1	1	1	1	1	1	1
16	0	0	0	0	0	0,02	0,22	0,98	1	1	1	1	1	1
24	0	0	0	0	0	0	0	0,02	0,06	0,63	1	1	1	1
32	0	0	0	0	0	0	0	0	0	0	0,63	1	1	1
40	0	0	0	0	0	0	0	0	0	0	0	0,63	1	1
56	0	0	0	0	0	0	0	0	0	0	0	0	0,63	1

Tabulka 9.5: Vyčíslení pravděpodobnosti úspěchu útoku hádáním $P_g(\alpha)$ pro různý počet hádaných bitů adresy N a různý počet opakování α

$$P_b(\alpha) = \alpha \cdot 2^{-N}$$

Rovnost 9.2: Pravděpodobnost úspěchu útoku hruhou silou P_b při hádání N bitů adresy a α násobném opakování

hádaných bitů (N)	počet opakování útoku (α)													
	1	4	16	64	256	2^{10}	2^{14}	2^{18}	2^{20}	2^{24}	2^{32}	2^{40}	2^{56}	2^{64}
1	0,50	1	1	1	1	1	1	1	1	1	1	1	1	
2	0,25	1	1	1	1	1	1	1	1	1	1	1	1	
4	0,06	0,25	1	1	1	1	1	1	1	1	1	1	1	
8	0	0,02	0,06	0,25	1	1	1	1	1	1	1	1	1	
16	0	0	0	0	0	0,02	0,25	1	1	1	1	1	1	
24	0	0	0	0	0	0	0	0,02	0,06	1	1	1	1	
32	0	0	0	0	0	0	0	0	0	0	1	1	1	
40	0	0	0	0	0	0	0	0	0	0	0	1	1	
56	0	0	0	0	0	0	0	0	0	0	0	0	1	

Tabulka 9.6: Vyčíslení pravděpodobnosti úspěchu útoku hádáním $P_b(\alpha)$ pro různý počet hádaných bitů adresy N a různý počet opakování α

9.1.5 Implementace ASLR: RANDMMAP

RANDMMAP je implementace Address Space Layout Randomization, které je schopno znáhodňovat umístění jakýchkoliv mapování včetně anonymních mapování. Znáhodnění je dosaženo modifikací funkce `do_mmap()`, kde probíhá volba počátečních adres mapování. Znáhodnění jednotlivých mapování jsou nezávislá a proto pokud exploit ke svému chodu vyžaduje například vykonávání kódu na zásobníku a zároveň v oblasti sdílených knihoven, potom musí jeho autor uhádnout $24 + 16$ bitů [SJ04].

9.1.5.1 Požadavek na formát programu

Technika znáhodňování RANDMMAP funguje pouze pro spustitelné soubory, které jsou ve formátu PIE. Formát PIE dnes není samozřejmý. Ověření o jaký typ ELF souboru se jedná postačí příkaz `file <název-souboru>`. Je-li soubor ve formátu PIE, potom je v jeho výstupu obsažen řetězec **ELF 64-bit LSB shared object, x86-64**[1], v opačném případě **ELF 64-bit LSB executable, x86-64**[1].

Position Independent Executable (PIE) je formát ELF binárního souboru. Jedná se o formát umožňující relokaci jednotlivých oblastí procesu uložených v binárním souboru na náhodné adresy v rámci AS procesu. Tentýž formát je používán pro kompilaci dynamicky linkovaných (sdílených) knihoven. PIE je někdy též označován jako etdyn. Spustitelné soubory, které jsou ve formátu etexec-ELF lze znáhodňovat nanejvýš technikou RANDEXEC (viz kapitola 9.1.6). Do formátu ELF-etexec je zkompilována dnes většina programů. Tento formát neobsahuje dostatečné množství relokačních informací potřebných k jejich relokaci na jinou adresu v paměti, než tu v nich specifikovanou.

[1]

9.1.6 Implementace ASLR: RANDEXEC

RANDEXEC je **implementace ASLR** procesů vykonávajících programy uložené ve formátu etexec.

Metoda RANDMMAP poskytuje zcela vyhovující řešení ASLR. Jejím hlavním nedostatkem je požadavek na formát spustitelného souboru – etdyn – zmíněný v kapitole 9.1.5.1. Kompilace programu do formátu etdyn nemusí být vždy možná, například z důvodu nedostupnosti zdrojových kódů programu.

RANDEXEC je alternativou RANDMMAP, která nevyžaduje formát etdyn. RANDEXEC implementuje randomizaci VAS binárních souborů ve formátu etexec. Občas vyvolává falešné poplachy a byla vyvinuta pouze jako technologický pokus [SJ04].

9.1.6.1 Princip

Relokace spustitelného souboru, který pro relokaci nebyl navržen – etexec – nemůže dosahovat kvalit řešení. RANDEXEC pracuje na stejném principu jako SEGMEXEC:

1. Obsah spustitelného souboru se nahraje na adresu v něm uvedenou čili na standardní pozici, kde své nahrání daný soubor očekává.
2. V tomto umístění se kód označí jako nespustitelný.
3. Obsah spustitelného souboru se nahraje do AS procesu ještě jednou a to do náhodně zvoleného umístění v User Space stejnými postupy jako v rámci RANDMMAP.
4. V tomto umístění se kód označí jako spustitelný.
5. Při vykonávání se program standardně pohybuje po svých původních adresách (uvedených v binárním souboru).
6. Každý pokus o spuštění kódu vyvolá výpadek stránky – Page fault.
7. Výpadek stránky vyhodnocuje funkce do_page_fault() modifikovaná patchem PaX.
8. Oprávněný přístup do paměti je přesměrován na adresu kopie, která spustitelná je.

RANDEXEC ochrana není ve výchozím stavu zapnuta. Zapíná se zvlášť pro každý program, který jí má být chráněn, nástrojem `paxctl` (viz kapitola 11.4.2.2).

9.1.7 Implementace ASLR: RANDUSTACK

RANDUSTACK je implementací ASLR pro userspace-stack.

Každý proces má v User Space zásobník. Tento zásobník je povinný a je vytvářen během volání `execve()`. Volání `fork()` jej pouze kopíruje do AS potomka. Standardní umístění zásobníku je na konci User Space a proto zásobník následně roste směrem k nižším adresám. Každé vlákno má vlastní userspace-stack vytvořené v User Space voláním funkce `mmap()`. RANDUSTACK znáhodňuje umístění každého userspace-stack hned dvakrát:

[1] Platí v případě spustitelného souboru zkompilovaného pro procesorovou architekturu x86-64. V případě spustitelného souboru zkompilovaného pro jinou procesorovou architekturu se výstup příkazu `file` pochopitelně liší v jejím označení.

1. Ve funkci `do_execve()` v souboru `fs/exec.c` se vytváří dočasný ukazatel na zásobník. Zde PaX znáhodňuje bity 2-11, čímž dosahuje možného posunu počátku mapování v rámci AS o

$$2^{10} \cdot \text{velikost stránky} = 2^{10} \cdot 4\text{kB}$$

.

2. Druhá část posunu počátku mapování po AS procesu se odehrává v okamžiku zavolání funkce `setup_arg_pages()`. V této funkci se standardně mapují stránky získané pro userspace-stack v předešlém kroku do AS procesu. Standardně se konec zásobníku zarovnává na adresu definovanou v konstantě `STACK_TOP`. Page eXec project modifikuje tuto konstantu (umístěnou v `include/asm-i386/a.out.h`) o náhodný posun daný proměnnou `delta_stack` v bitech 12 až 27. Tento druhý krok zajistí, že díky RANDUSTACK je možné hýbat s userspace-stack po AS až o 256 MB.

9.1.8 Implementace ASLR: RANDKSTACK

Každý proces disponuje dvěma paměťovými stránkami v AS, ve kterých leží jeho kernelspace-stack. Tento zásobník slouží k podpoře obsluhy operací prováděných v kernel mode, jakými jsou systémová volání, hardwarová přerušení a výjimky vzniklé na CPU. Do kernelspace-stack se mimojiné ukládá adresa instrukce ležící v User Space, která inicializovala přepnutí kontextu z user mode do kernel mode. Ukazatel na kernelspace-stack může být na rozdíl od ukazatele na userspace-stack randomizován při každém přepnutí kontextu [SJ04], neboť vždy vzniká „z ničeho“ čili prázdný. To umožňuje jej randomizovat při každém přepnutí kontextu znovu a nikoliv pouze jednou, jako je tomu v případě userspace-stack při spuštění procesu/vlákna. Posun, který PaX umožňuje pro kernelspace-stack je 128 B.

9.2 Důsledné vynucení nespustitelné paměti

Množící se útoky spadající do kategorie 8.1.2 přiměly návrháře procesorových architektur k tomu, aby do svých děl zakomponovali podporu pro nespustitelnou paměť. Dnes již je na většině procesorových architektur podpora pro nespustitelné stránky přítomna, stejně tak jako i v Linuxu. Vzhledem k tomu, že

- Linux je co se týče ne/spustitelnosti stránek nedůsledný a
- architektura IA-32 podporu ne/spustitelných stránek postrádá,

existuje zde prostor pro PaX, který se snaží výše uvedené napravit, což činí v rámci techniky pojmenované NOEXEC.

9.2.1 Vylepšení ne/spustitelné paměti poskytované PaXem

Zatímco podpora nespustitelných stránek (např. v x86-64) hardwaru je již přítomna, stávající programové vybavení – jádro – ji zpravidla **používá nedůsledně**[2]. Například distribuční jádro ve verzi 3.16 používané v Debian Jessie

- **ochranou proti spuštění vybaví pouze některá mapování v AS procesu** a

  Userspace-stack učiní nespustitelným, nicméně halda, jejíž spustitelnost představuje taktéž zranitelnost, byť o něco obtížněji zneužitelnou, ponechá spustitelnou.
- **ponechá útočníkovi přístupné možnosti tuto ochranu deaktivovat**.

  Útočník má možnost modifikovat přístupová oprávnění paměťové stránky voláním funkce `mprotect()` čili má možnost učinit jakoukoliv stránku dodatečně spustitelnou, podaří-li se mu zavolat `mprotect()`.

Dodatečná softwarová podpora nespustitelných stránek na x86-64 v podobě PaX tak má stále potenciál ochranu před nespustitelnými stránkami učinit výrazně silnější.

9.2.1.1 Vynucení ne/spustitelnosti na „jakémkoliv“ mapování (NOEXEC)

NOEXEC je technika ochrany proti útokům typu 8.1.2 vedeným skrze modifikaci v AS již existujícího mapování. Stejně tak, jako i v jiných oblastech bezpečnosti, i zde se uplatňuje princip minimálních privilegií čili jsou udělována minimální privilegia, která postačují ke správné funkci programu. Pakliže proces nevyžaduje ke svému chodu generování spustitelného kódu, neměl by jej být chopen. NOEXEC

- **vynutí podporu pro nespustitelné stránky v jádře** a

  Anonymní mapování budou již výhradně nespustitelná.
- **vynutí zohledňování požadavků specifikovaných v hlavičce ELF spustitelných souborů**.

  Spustitelné soubory ve formátu ELF budou mapovány do AS se spustitelným vždy jen tím ELF segmentem, který obsahuje programový kód. Ostatní ELF segmenty budou mapovány do AS jako nespustitelné, nebude-li v rámci ELF souboru řečeno jinak.

[2] Využití přítomnosti NX bitu se pochopitelně liší dle použitého jádra.

Výše zmíněná opatření by byla k ničemu bez záruky, že během vykonávání programu nedojde ke změně parametrů jednotlivých mapování (viz následující kapitola).

9.2.1.2 Znemožnění dodatečné modifikace ne/spustitelnosti mapování (MPROTECT)

Oprávnění paměťových stránek lze obecně určit

- voláním `mmap()` při umísťování jednotlivých mapování do AS procesu či
- voláním `mprotect()`.

Tato volání jsou samozřejmě dostupná i útočníkovi. Aby technika NOEXEC fungovala, nesmí být možné paměťovým stránkám měnit jejich oprávnění. Proto se oprávnění jednotlivých paměťových stránek zamykají. Ochrana paměťových stránek *MPROTECT*

- **zabrání vytvoření mapování, které by bylo zapisovatelné a spustitelné zároveň**,
- **anonymní mapování učiní vždy pouze nespustitelným** a
- **zakáže přidělení spustitelnosti mapování, které dříve bylo či je zapisovatelné**.

Na systémech, kde je aktivována ochrana MPROTECT spuštění programu, jehož kód je uveden v C kódu 9.2 skončí neúspěchem. Příčinou selhání bude volání funkce `mmap()` čili v daném případě pokus o alokaci 1 kB velkého úseku paměti s právy zápisu i spouštění jeho obsahu, což MPROTECT ochrana nedovoluje.

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/mman.h>
#include <errno.h>
#include <string.h>

int main()
{
    size_t* m;

    m = mmap( 0, 1024, PROT_WRITE | PROT_EXEC, MAP_PRIVATE | MAP_ANONYMOUS, -1, 0 );

    if( m == MAP_FAILED )
    {
        printf("mmap failed: %s\n", strerror(errno));
        return 1;
    }
    printf("mmap succeeded: %p\n", m);
    return 0;
}
```

C kód 9.2: Zdrojový kód programu, jež v systémech s aktivovaným MPROTECT nemůže skončit zdarem

9.2.2 Přidání ne/spustitelné paměti do IA-32 systémů

Záznamy v tabulce stránek v IA-32 nenesou informaci o ne/spustitelnosti stránky a proto MMU nemá informaci o tom, zda-li obsah nesený danou stránkou je přípustné spouštět či nikoliv. Naopak architektury jako x86-64 či ARMv6 podporu pro nespustitelné stránky v MMU mají, neboť v záznamu jejich tabulky stránek je vždy přítomen NX bit – příznak nespustitelnosti [NXexp]. Výrobci procesorů podporu ne/spustitelné paměti označují různě –

- Intel jako *eXecute Disable (XD)*,
- AMD jako *Enhanced Virus Protection (EVP)* a
- ARM jako *eXecute Never (XN)*.

PaX přidává na IA-32 softwarovou emulaci ne/spustitelných stránek a to hned ve dvou možných variantách:

9.2.2.1 Implementace založená na stránkování (PAGEEXEC)

PAGEEXEC je jednou ze dvou technik **emulace ne/spustitelných stránek** pro procesorovou architekturu IA-32, které PaX nabízí. Tato emulace je postavena na existenci dvou TLB. Více TLB jednotek (viz kapitola 4.4.1.3) je v x86-kompatibilních procesorech standardem již od procesorů i586. Zavedeny v nich byly

- **instrukční TLB (ITLB)** a

TLB, do které se ukládají překlady nejčastěji přistupovaných paměťových stránek, k nimž je přistupováno s úmyslem je vykonávat. Předpokládá se, že tyto paměťové stránky obsahují strojové instrukce.

- **datová TLB (DTLB).**

  TLB, do které se ukládají informace o nejčastěji přistupovaných stránkách, k nimž je přistupováno s úmyslem je nevykonávat. Předpokládá se, že tyto paměťové stránky obsahují data zpracovávaná programem.

Princip fungování PAGEEXEC je následující:

1. **Všechny stránky VAS, které mají být nespustitelné, mají v tabulce stránek nastaven buďto tzv. supervisor bit nebo „non present bit“.**

   Supervisor bit (SB) je jedním z příznaků vedených při každém záznamu tabulky stránek, jehož případné nastavení zamezí každému přístupu k dané paměťové stránce s úrovní oprávnění Ring 1 a nižší čili zamezí každému přístupu ke stránce, není-li veden z Ring 0 (viz kapitola 5.1.1) čili kernel mode [Li]. V user mode se ke stránce s nastaveným SB dostat nelze.

2. **CPU chce přistoupit na konkrétní LA adresu[3] v paměti.**

   CPU disponuje LA adresou, která je pochopitelně k ničemu bez znalosti jejího mapování do fyzického adresního prostoru. Proto MMU započne hledání jejího „zobrazení“ do LAS.

3. **Překlad LA → VA MMU nalezne v tabulce segmentových deskriptorů.**

   Nyní zbývá nalézt zobrazení VA → PA.

4. **Jelikož přístup do tabulky stránek je drahý, <PEKNE-BY-BYLO-CITOVAT> MMU nejprve ověřuje přítomnost překladu LA v příslušné TLB, která odpovídá typu přístupu do paměti.**
   - Překlad je v ITLB/DTLB přítomen.

     Překlad LA na PA byl v historii běhu programu již proveden a s ním proběhla i kontrola přístupových oprávnění. Je-li překlad již přítomen v TLB, potom je **přístup povolen**.
   - Překlad není v ITLB/DTLB přítomen.

     V takovém případě bude prohledávána tabulka stránek (viz bod 5). Nebude-li nalezen překlad, čili nebude-li stránka požadované LA existovat či nebude-li přístup ke stránce v souladu s přístupovými oprávněními, bude vržena výjimka Page fault.

5. **Přístup ke stránkám s nastaveným SB z kódu běžícím s oprávněním nižším nežli je Ring 0 vyvolá výjimku Page fault.**

   Při přístupu do stránky paměti z kódu běžícím s oprávněním Ring 0, čili běžícím v kernel mode, je podmínka nutná pro přístup ke stránce – supervisor bit – splněna a k vržení výjimky nedochází.

6. **V rutinách obsluhujících tuto výjimku se vyhodnotí, zda-li přístup k paměti, který výjimku vyvolal byl veden z DTLB či ITLB.**
   - Přístup byl veden skrz DTLB.

     Příčinou přístupu do paměti nebyl úmysl vykonat její obsah. Není důvod procesoru bránit v načtení.
     (a) Supervisor bit je odnastaven.
     (b) Překlad VA → PA je nahrán do DTLB.
     (c) Supervisor bit je nastaven.
   - Přístup byl veden skrz ITLB.

     Příčinou přístupu do paměti byl jednoznačně úmysl procesu načíst instrukci. Při obsluze výjimky Page fault jsou porovnávány
     – adresa instrukce, při jejímž načítání došlo k vržení výjimky a
     – adresa paměti, na které došlo k vržení výjimky.
     Pokud se výše uvedené adresy shodují, výjimka Page fault byla zapříčiněna PaXem – v tomto případě jeho ochranou proti spouštění nespustitelných stránek PAGEEXEC.

Technika PAGEEXEC tedy v podstatě jen „přetěžuje význam supervisor bitu“ v tabulce stránek, kde tento bit zaručuje „nespustitelnost stránky“ pro jakýkoliv přístup k ní vedený v user mode čili lépe řečeno s oprávněním nižším nežli Ring 0. Datové přístupy PAGEEXEC nechává bez povšimnutí, neboť v takovém případě je SB automaticky pro přistupovanou stránku na dobu nezbytně nutnou odnastaven.

Technika PAGEEXEC existuje též ve variantě, kdy v tabulce stránek není přetěžován supervisor bit, nýbrž „non present bit“. Tato možnost je však v praxi nepoužívá, neboť instrukční přístupy vedené z Ring 0 způsobují Page fault také, čímž dále systém zpomalují.

[3] Čtenář byl s faktem, že CPU pracuje s logická adresa (LA) obeznámen již v kapitole 4.1.1.

9.2.2.2 Implementace založená na segmentaci (SEGMEXEC)

SEGMEXEC je jednou ze dvou technik emulace ne/spustitelných stránek pro procesorovou architekturu IA-32, které PaX nabízí. Tato technika emulace využívá toho, že

- **x86-kompatibilní procesory podporují segmentaci zatímco GNU Linux ji „nevyužívá“ (viz kapitola 4.5)**,
- **pro každý přístup do paměti se provádí dvoufázový překlad adres** a

  Popsán v kapitole 4.1.
- **Linux segmentaci v podstatě nepoužívá**.

  Přesněji řečeno Linux segmentaci využívá, avšak pouze tím způsobem, aby „mu nepřekážela“ čili všechny segmenty začínají na téže adrese a plně se překrývají [03]. Tím se eliminuje význam komponenty LA zvané selektor segmentu. Druhá komponenta LA – offset v rámci segmentu – se tak stane jediným prvkem LA nesoucím informaci a v podstatě splyne s VA a jelikož obory zobrazení LA $\rightarrow$ VA se tak stanou shodnými, lze toto zobrazení nazvat identickým.

Celý aparát nachystaný v x86-kompatibilních procesorech a Linuxu pro podporu segmentace **lze znásilnit pro realizaci jiných, původně nezamýšlených, cílů**. Jedním z nich je právě emulace nespustitelných stránek dodávaná PaXem pod názvem SEGMEXEC.

Princip fungování SEGMEXEC je následující:

1. **US procesu je rozdělen na dva stejně velké segmenty**.
   - **Datový segment SEGMEXEC (SE-DS)** a

     Část US pokrývající AS s nižšími adresami[2] určená pro uložení mapování pro datový přístup.
   - **Kódový segment SEGMEXEC (SE-KS)**.

     Část US pokrývající AS s vyššími adresami[3] určená pro uložení mapování pro instrukční přístup.
2. **Všechny stránky původně rozeseté po celém US jsou umístěny výhradně v SE-DS.**

   US procesu je tím omezen na ½ své původní velikosti.
3. **Stránkám, jejichž obsah má být spustitelný, jsou vytvořeny kopie v SE-KS**.

   Tyto kopie jsou mapovány do AS takovým způsobem, aby byly vždy uložené na adrese složené z adresy v SE-DS a pevně daného offsetu o velikosti poloviny[4] US. Přičtení offsetu je spolehlivě posune do segmentu SE-KS.
4. **Program standardně přistupuje do SE-DS.**

   Všechen obsah US je umístěn v SE-DS a proto přistupovaná adresa patří vždy do SE-DS.
5. **Datové a instrukční přístupy jsou rozlišeny při překladu**.

   Za normálních okolností všechny přístupy do paměti podléhají témuž zobrazení LA $\rightarrow$ VA, neboť datový i kódový segment mají tentýž offset rovný hodnotě 0:

   $$(\text{<datový segment>},\text{<offset>}) \rightarrow 0 + \text{<offset>} \tag{9.1}$$
   $$(\text{<kódový segment>},\text{<offset>}) \rightarrow 0 + \text{<offset>} \tag{9.2}$$

   Technika SEGMEXEC způsobí následující:

   $$(\text{<datový segment>},\text{<offset>}) \rightarrow 0 + \text{<offset>} \tag{9.3}$$
   $$(\text{<kódový segment>},\text{<offset>}) \rightarrow \tfrac{1}{2}\,\text{AS} + \text{<offset>} \tag{9.4}$$
6. **Instrukční přístup směřující k nespustitelné stránce vyvolá Page fault výjimku**.

   Instrukční přístup k nespustitelné stránce je mechanismem segmentace přesměrován z SE-DS do SE-KS, kde se však přistupovaná stránka nenachází. To způsobí výjimku Page fault. Obsluha této výjimky je modifikována tak, že dojde k závěru, že se jednalo o instrukční přístup k nespustitelné stránce a proces typicky ukončí.

Technika SEGMEXEC tedy v podstatě jen využívá aparát segmentace k emulaci podpory nespustitelných stránek.

[1] Velikost US je při použití architektury IA-32 3 GiB.
[2] Lineární adresní prostor 0 - 1.5 GiB.
[3] Lineární adresní prostor 1.5 - 3 GiB.
[4] V případě architektury x86-32 je offset přičítaný k původní adrese délky 1.5 GiB.

9.2.2.3 Dopady emulace ne/spustitelných stránek

Nemá smysl se rozepisovat o pozitivních dopadech emulace ne/spustitelných stránek na architektuře, která by jinak než emulací na ne/spustitelné stránky nedosáhla. Zajímavější je porovnání dvou PaXem poskytovaných technik emulace co se týká jejich negativních vedlejších účinků na běh systému. Nabízí jej tabulka 9.7.

Dopady při nasazení PAGEEXEC	Dopady při nasazení SEGMEXEC
Spotřeba paměti nevzroste. Na rozdíl od SEGMEXEC nezmenší PAGEEXEC VAS, jež má aplikace k dispozici.	Velikost využitelného VAS je nasazením SEGMEXEC změnšena o polovinu[4]. Navíc spotřeba paměti vzroste, neboť některé oblasti VAS jsou v paměti uloženy dvakrát.
Růst spotřeby procesorového času. Vykonávání programu je zpomaleno v důsledku častějšího vyvolávání Page fault výjimky.	Překlad adresy při instrukčním přístupu z adresy v SEDS na adresu v SE-KS je prováděn automaticky a nespotřebovává procesorový čas.

Tabulka 9.7: Porovnání dopadů emulací ne/spustitelných stránek pro IA-32

9.3 Nefunkčnost programů zapříčiněná PAGEEXEC/SEGMEXEC

Emulace NOEXEC není řádné NOEXEC. Emulace ze své podstaty musí mít nežádoucí účinky. Vyjma spotřeby určitého množství procesorového času s sebou emulace NOEXEC čili nasazení PAGEEXEC či SEGMEXEC přináší i další nechtěné efekty. Jedním z nich může být znemožnění správného chodu programů, které si generují části svého strojového kódu za běhu[5].

9.3.1 Trampolíny

Trampolína je posloupnost strojových instrukcí generovaná v reálném čase (tzn. za běhu procesu) procesem samotným a umisťovaná do userspace-stack. Proces ji generuje, neboť díky ní zpřístupní lokální proměnné deklarované v rámci jedné RJ v jiné RJ. Na zápisu sekvence strojových instrukcí na zásobník není nic závadného, a to ani pro PAGEEXEC/SEGMEXEC. Pokus o spuštění této sekvence je však již těmito technikami vyhodnocen jako pokus o spuštění nespustitelného obsahu. Pro představu v ukázce C kódu 9.3 by proces generoval trampolínu – několik málo instrukcí strojového kódu – za účelem zpřístupnění proměnné `i` ve funkci `vypis()`, která je v daném případě volána mimo funkci `main()`. Vnořená funkce `vypis()` počítá s existencí lokální proměnné `i` na zásobníku a bez trampolíny se k ní nemá jak dostat.

9.3.1.1 Standardní řešení trampolín

1. **V AZ funkce `main()` se vymezí prostor na userspace-stack, do kterého se vloží instrukce dané procesorové architektury představující**
   - přesun hodnoty z registru EBP do registru ECX a
   - skok na funkci, jež vyžaduje přístup k lokálním proměnným jiné funkce (v C kódu 9.3 se jedná o funkci `vypis()`).

   Tyto dvě instrukce spouštěné ze zásobníku se nazývají trampolína.
2. **Ve funkci `proved()` dojde k provedení těchto dvou instrukcí.**
3. **K zpřístupnění lokálních proměnných funkce `main()` dojde prostřednictvím registru ECX.**

9.3.1.2 Nestandardní řešení trampolín (EMULTRAP)

EMULTRAP je technika poskytovaná PaXem, která použití trampolín umožní i v případě aktivního PAGEEXEC/SEGMEXEC. Aktivovat EMULTRAP lze na úrovni jednotlivých ELF spustitelných souborů pomocí značkování (viz kapitola 9.11). EMULTRAP emuluje vybrané (nejčastější/nejrozšířenější) trampolíny, čili generuje určitý strojový kód procesoru, z čehož vyplývá, že EMULTRAP je architekturně závislá. Implementace EMULTRAP zatím existuje pouze pro architekturu IA-32.

9.4 Značkování spustitelných souborů souborů pro PaX

V současnosti **existují dvě metody, jak značkovat ELF soubory** za účelem informování PaXu, jak má s nimi naložit:

[5] Vykonávání za běhu generovaného strojového kódu je vyžadováno např. při využívání vložených funkcí (tzv „nested functions“) v jazyce C nebo některými JIT překladači.

```c
#include <stdio.h>

typedef void (*fn_t) (void);

void proved(fn_t fn)
{
    (*fn)();
}

void main()
{
    int i;
    scanf("%d", &i);

    void vypis()  // Vnořená definice funkce (podporováno překladačem GCC).
    {
        printf("i=%d\n",i);
    }

    proved(f2); // nastavení trampolíny pro vnořenou funkci f2()
    return 0;
}
```

C kód 9.3: Situace, kdy se za běhu programu vygenerují instrukce pro zprístupnění proměnné `i`, tzv. trampolína

9.4.1 Značkování PT_PAX

Technika značkování *PT_PAX* ukládá značky přímo do ELF hlavičky zvané „PAX_FLAGS“ obsažené přímo v ELF souboru. Výhodou tohoto řešení je, že značky zůstávají pevně svázány s daným ELF souborem vždy a to i v případě, je-li kopírován. Nevýhodou tohoto přístupu je, že program musí disponovat ELF hlavičkou „PAX_FLAGS“, aby vůbec mohl být spuštěn. Většina linuxových distribucí tuto hlavičku nemá ve svých spustitelných souborech a knihovnách přítomnu a její dodatečné přidání je problematické. Použití PT_PAX je tak proto nedoporučováno, nicméně stále se jedná o podporovanou formu značkování spustitelných souborů.

9.4.2 Značkování XATTR_PAX

Technika značkování *XATTR_PAX* umisťuje značky do rozšířených atributů souboru. Výhodou tohoto řešení je, že není nutné nijak modifikovat původní ELF soubory. Nevýhodou je, že značky nejsou pevně svázány se spustitelným souborem, nýbrž jsou v uložené mimo něj v rámci FS. To přináší požadavky na nástroje, které s danými soubory manipulují – kopírují je, přesouvají je, archivují je, atp. Pro zachování nastavení značek musí nástroje manipulující se soubory zohledňovat existenci rozšířených atributů souborů (viz kapitola 6.5.2).

9.5 Obrana proti útoku ret2usr (UDEREF a KERNEXEC)

Jako obrana proti útokům popisovaným v kapitole 8.9.2 by úplně stačilo, kdyby jádro nemohlo přistupovat nestandardními cestami do User Space čili kdyby každý přístup do User Space musel být zprostředkováván funkcemi k tomu určenými – `copy_to_user()`, `copy_from_user()` atp. Tento požadavek lze splnit a plní jej hned několik subjektů –

- patch Grsecurity/PaX v podobě volby KERNEXEC (viz kapitola 9.5.1),
- patch Grsecurity/PaX v podobě volby UDEREF (viz kapitola 9.5.2),
- Intel v podobě technik SMAP/SMEP (viz kapitola 9.5.3),
- ARM technikou Privileged Execute-Never (PXN),
- kGuard [KPK12]
- a zřejmě i řada dalších.

Princip všech výše vyjmenovaných technik obrany proti útoku ret2usr vždy spočívá v oddělení paměťových prostorů Kernel Space a User Space takovém, aby nebylo bezpodmínečně možné dereferencovat ukazatel směřující do User Space. **Jmenované ochrany proti útoku ret2usr ať už v podání patche Grsec/PaX či jako SMAP/SMEP nejsou, bohužel, vše spásné, neboť byla vynalezena vylepšená verze ret2usr zvaná *Return to direct mapped memory (ret2dir)*, proti níž výše zmíněné ochrany nezafungují [PKK14].**

9.5.1 Technika KERNEXEC

KERNEXEC je bezpečnostní technika, která

- datové stránky Kernel Space činí nespustitelnými a
- některé významné struktury jádra jako jsou např. IDT, GDT, vybrané komponenty tabulky stránek či tabulku systémových volání činí nezapisovatelnými.

9.5.1.1 Implementace KERNEXEC na x86-32

Na architektuře x86-32 technika KERNEXEC svého cíle dosahuje segmentací. Kernel Space je mapován do 1 GB velkého segmentu. Pokud se kód umístěný do tohoto segmentu pokusí načíst data/instrukce z User Space, potom je vržena výjimka Page fault. Principem implementace je tedy **využití, Linuxem jinak nevyužívaného, mechanismu segmentace hlavní paměti**. Nasazením segmentace vzniká problém s chybným předpokladem jádra, že logická a lineární adresa splývají. Je proto zapotřebí provádět mezi nimi překlad.

9.5.1.2 Implementace KERNEXEC na x86-64

Na architektuře x86-64 technika KERNEXEC svého cíle dosahuje vynucením používání NX bitu respektive upravením přístupových oprávnění vedených pro jednotlivá mapování jádra.

9.5.2 Technika UDEREF

UDEREF je technika, která zabraňuje neúmyslnému[6] přístupu do User Space z jádra. Konkurencí jí je technika SMAP uvedená v procesorech Intel Haswell (více viz kapitola 9.5.3). Zabrání dereferencování ukazatelů ukazujících do User Space v situacích, kdy jsou přípustné pouze ukazatele ukazující do Kernel Space. K jejich rozlišení použije mechanismus segmentace [Luk14].

9.5.2.1 Implementace UDEREF

UDEREF každý vstup vykonávání z kódu uživatelského programu do kódu jádra obohatí o následující:

- Bezprostředně po vstupu do kernel mode se přemapují (virtuálně přesunou) všechna mapování User Space uživatelského procesu do jiné oblasti v paměti, kde budou označena jako nespustitelná.
- Pokusy o vykonání dat z této oblasti v kernel mode vyvolají výjimku.
- Před výstupem z kernel mode se obsah User Space přemapuje do původní jeho pozice.

Tímto opatřením UDEREF sice neznemožní dereferencovat ukazatele směřující do User Space obecně, nicméně **dosáhne nevykonavatelnosti obsahu**, na nějž tyto ukazatele ukazují. Jediným povoleným přístupem z kernel mode do User Space zůstanou funkce k tomu prvoplánově určené.

9.5.3 Technika SMAP/SMEP

Supervisor Mode Execution Protection neboli *SMEP* je technika zabraňující kódu běžícímu v Ring 0/kernel mode v instrukčních přístupech do US [Kit+15]. Poprvé byla přítomna v procesorech Intel IvyBridge. Na rozdíl od KERNEXEC není tato technika schopna zabránit modifikaci důležitých datových struktur jádra.

Supervisor Mode Access Protection neboli *SMAP* je technika zabraňující kódu běžícímu v Ring 0/kernel mode v datových přístupech do US [Kit+15]. Poprvé byla přítomna v procesorech Intel Haswell. Obě tyto ochrany při přístupu k paměťové stránce ležící v User Space vyvolají Page fault výjimku. **Nejedná se o součásti PaXu, nicméně obě souvisí s ochranou proti útokům typu ret2usr/ret2dir.**

Jedná se o rozdělení paměťového prostoru, které je podpořené hardwarem, tudíž implementace je závislá na výrobci procesoru a funkčnost těchto technik vyžaduje doplnění podpory do jádra. Ta do něj byla pro x86 procesory Intelu přidána v jeho verzi 3.7.

9.5.4 Technika XPFO

Proti útoku ret2dir je účinnou ochranou sada záplat jádra zvaná *eXclusive Page Frame Ownership (XPFO)* [PKK14]. V citovaném dokumentu je též popsán princip jejich fungování. **Nejedná se o součást PaXu, nicméně souvisí s ochranou proti útokům typu ret2usr/ret2dir.**

XPFO čili „vynucení exkluzivního vlastnictví stránek“ odebírá jádru možnost přístupu k paměťovým stránkám prostřednictvím přímého mapování. Princip této techniky je následující:

1. Je alokována stránka v User Space.
2. Standardně je během této alokace vytvářeno přímé mapování na tuto stránku vedoucí.
3. Přítomnost XPFO vzápětí způsobí odstranění tohoto mapování.
4. Stránka v User Space je dealokována.

[7] Volba UDEREF tedy nezabraňuje přístupu do User Space obecně. Výjimka je udělena funkcím jádra pro tuto agendu prvoplánově určeným – `copy_to_user()`, `copy_from_user()` atd.

Útočník bez existence záznamu o stránce v přímém mapování nebude moci skrze adresu přímého mapování k této stránce přistoupit. Jakýkoliv přístup skončí kvůli nedostatku přístupových oprávnění.

9.6 Volba PAX_SOFTMODE

Veškeré ochrany konfigurované v kapitole 9 budou standardně vypnuty, nicméně bude je možné aktivovat uvedením patřičných příznaků v rámci GrSecurity ACL. Zapnutí této volby vyžaduje též zapnutí podpory pro PT_PAX_FLAGS nebo XATTR_PAX_FLAGS, aby bylo možno značkovat spustitelné soubory.

9.7 Volba PAX_EI_PAX

Aktivuje starší způsob uložení příznaků ovlivňujících činnost PaX – do ELF hlaviček. Příznaky nesou režijní informace PaXu:

- Vynucovat nespustitelné stránky? Ano. / Ne.
- Použitá realizace nespustitelných stránek? RANDMMAP / RANDEXEC

Zapnutí volby PAX_EI_PAX umožní pozdější určení voleb PaXu pro jednotlivé spustitelné soubory nástrojem chpax.

9.8 Volba PAX_PT_PAX_FLAGS

Příznaky ovlivňující chování PaX nad subjekt budou ukládány do speciální hlavičky v rámci ELF formátu. Tato implementace podporuje soft-mode běh PaX zmíněný v kapitole 9.6. Jedná se výchozí způsob uložení.

9.9 Volba PAX_MPROTECT

Znepřístupní aplikacím možnost si voláním funkce mprotect() vypnout některé části mechanismu vynucování nespustitelných stránek. Volání mprotect() totiž standardně umožňuje stanovit oprávnění k bloku paměti. Tato volba by v případě aktivace volby PAX_NOEXEC rozhodně měla být zapnuta.

9.10 Volba PAX_ELFRELOCS

Aktivace voleb

- PAX_NOEXEC a
- PAX_MPROTECT

představuje účinnou prevenci před vykonáváním útočníkem vloženého kódu – shellcode. Útočníkovi po jejich aktivaci zbývají ještě dvě příležitosti ke spuštění kódu:

ve VAS procesu již existující kód a
: Vykonávání kódu v oblastech jakými jsou text a MMS je pro správné fungování programu nevyhnutelné a proto jej nelze omezovat.

kód do VAS procesu dodatečně připojený voláním mmap().
: K účelu převzetí kontroly nad procesem potřebuje útočník knihovnu, která **není** ve formátu *PIC*.

Position Independent Code (*PIC*) je ELF formát souboru připravený na relokace v rámci VAS procesu.

Standardně Grsec neumožňuje relokovat ELF kód, respektive volba PAX_ELFRELOCS je vypnuta. Při načítání sdílených knihoven do VAS dochází k relokacím **vždy**. Zákaz tohoto typu relokací vyžaduje, aby jej žádná v systému přítomná knihovna nevyžadovala. Toho lze dosáhnout kompilací všech knihoven do PIC formátu.

Pro vynucení PIC formátu v gcc existuje přepínač -fpic. V případě, že se v systému vyskytují knihovny, které nejsou v PIC formátu, nemusí systém vůbec naběhnout. Formát dynamicky linkované knihovny zobrazí příkaz 9.3.

```
readelf -S <lib-name>
```

Příkaz/y 9.3: Příkaz, z jehož výstupu lze vyčíst informaci o formátu knihovny. Je-li v něm přítomen řetězec .rel.text, potom zkoumaná knihovna **není** v PIC formátu

9.11 Volba PAX_SOFTMODE

Techniky existující v rámci PaXu, které umožňují své zapnutí/vypnutí za běhu systému, mohou být vynucovány buď-to

- **implicitně pro všechny spustitelné soubory v systému** nebo

  Omezení daná PaXem jsou v tomto režimu aktivní pro veškeré spustitelné soubory. Jednotlivá omezení lze pro jednotlivé spustitelné soubory **vypínat**.
- **pouze pro explicitně vyjmenované spustitelné soubory**.

  Tento režim běhu PaXu je označován jako *SOFTMODE*. Ochrany poskytované PaXem se v tomto případě musí explicitně **zapínat** pro jednotlivé spustitelné soubory.

Spustitelné soubory se vyjímají/zahrnutí z/do vlivu opatření PaXu pomocí voleb – přepínačů – uvedených v tabulce 9.8. Tyto přepínače jsou načítány linuxem při zavádění příslušného ELF souboru do paměti. Tímto způsobem se zpravidla redukují restrikce PaXu, které znemožňují chod programu v NOSOFTMODE. Problém nastává ve chvíli, vyžaduje-li výjimku z opatření PaXu knihovna, kterou si ELF soubor dynamicky linkuje.

Přepínač vyjmutí (v NOSOFTMODE)	Přepínač zahrnutí (v SOFTMODE)	Dopad uvedení přepínače na spustitelný soubor
`-p`	`-P`	Aktivace, případně deaktivace techniky PAGEEXEC pro daný spustitelný soubor.
`-e`	`-E`	Aktivace, případně deaktivace techniky EMULTRAP pro daný spustitelný soubor.
`-m`	`-M`	Aktivace, případně deaktivace techniky MPROTECT pro daný spustitelný soubor.
`-r`	`-R`	Aktivace, případně deaktivace techniky RANDMMAP pro daný spustitelný soubor.
`-s`	`-S`	Aktivace, případně deaktivace techniky SEGMEXEC pro daný spustitelný soubor.

Tabulka 9.8: Přepínače používané pro vyjmutí/zahrnutí spustitelného souboru do PaXu

9.12 Volba PAX_PAGEEXEC

Grsecurity/PaX nabízí hned dvě možné implementace nespustitelných stránek:

- PAGEEXEC (viz kapitola 9.2.2.1) a
- SEGMEXEC (viz kapitola 9.2.2.2).

V případě nenastavení volby `PAX_PAGEEXEC` bude aktivována implementace SEGMEXEC. Prostředky spotřebované PAGEEXEC implementací nespustitelných stránek jsou platformně závislé:

Vysoké nároky x86-32 bez NX bitu.
: Emulace NX bitu na procesorech starších Pentia 4 včetně je dopad na výkon natolik citelný, že se zapínání volby `PAX_PAGEEXEC` v tomto případě nedoporučuje [GrSecDoc].

Nízké nároky na x86-32 s NX bitem, x86-64, sparc, sparc64, ...
: Dopad na výkonnost systému je zanedbatelný až žádný [GrSecDoc].

9.13 Volba PAX_MEMORY_SANITIZE

Jádro smaže obsah paměťových stránek v okamžiku jejich uvolnění z paměti. Proces, jehož paměťové stránky se namapují na tentýž prostor tak nedostane příležitost číst data, k nimž nemá a nikdy neměl přístupová oprávnění. Tato volba je užitečná zejména pro procesy, jejichž bytí je krátké. Pro dlouho-běžící také, ale pouze v případě, že se paměti s citlivými/zneužitelnými daty rychle zbavují. Data v „živých“ stránkách volba nijak nepostihne. To znamená, že obsah stránek, které jsou jen odstránkovány do odkládacího prostoru, zůstane.

9.14 Volba PAX_MEMORY_STACKLEAK

Jádro smaže kernelspace-stack před návratem z obsluhy systémového volání. Tímto zredukuje životnost dat ležících na kernelspace-stack. Toto opatření zmenší množství informací, které mohou uniknout [Luk14]. Původně byla tato ochrana implementována voláním fn `memset()` na celý kernelspace-stack (cca 8 kB). Toto řešení bylo příliš pomalé. Volání `memset()` nyní neprobíhá nad celým kernelspace-stack, nýbrž pouze nad jeho částí a to není dokonalé. Řešení, které přinesl PAX_MEMORY_STACKLEAK spočívá v existenci dvou kernelspace-stack, z nichž jeden je používán pro kopírování hodnoz do/z kernelspace a druhý pro všechno ostatní. Chyba v jádře samotném může stále způsobit únik informací, které byly vloženy do kernelspace-stack současným systémovým voláním. Předchozí systémová volání však díky volbě PAX_MEMORY_STACKLEAK již nejsou viditelná [GrSecDoc]. Tato volba je prevencí útoku uvedeného v kapitole 8.6.

9.15 Volba PAX_MEMORY_STRUCTLEAK

Volba STRUCTLEAK vynutí inicializaci všech proměnných jádra označených atributem `__user` nulovou hodnotou. Případný jejich výskyt na kernelspace-stack tak po aktivaci volby STRUCTLEAK již nepředstavuje zranitelnost. Tato technika byla psána původně proti zranitelnosti CVE-2013-2141. Tato zranitelnost umožňuje neoprávněné získání hodnot proměnných jádra zneužitelné k dalším útokům (více o zranitelnosti viz [CVE13]). Hodnoty proměnných, kterých se dané opatření týká, se k uživateli čili do User Space dostávají naprosto standardní cestou – systémovými voláními. V případě CVE-2013-2141 se jedná o systémová volání `tkill()` či `tgkill()` [CVE13]).

Funkcionalita poskytovaná volbou PAX_MEMORY_STRUCTLEAK byla do jádra přidána v jeho verzi 4.11 pod volbou GCC_PLUGIN_STRUCTLEAK.

9.16 RAP

Return Address Protection (RAP) je **technika obrany proti útoku ROP**. Aplikace RAP modifikuje [Pol16]

- prolog všech funkcí z podoby uvedené ve strojovém kódu x86 9.1 do podoby uvedené ve strojovém kódu x86 9.2 a
- epilog všech funkcí z podoby uvedené ve strojovém kódu x86 9.3 do podoby uvedené ve strojovém kódu x86 9.4.

Principem ochrany vůči ROP v podání RAP je obohacení prologů a epilogů všech funkcí o kód ověřující skutečnost, že během vykonávání funkce nebyla na vrchol zásobníku podstrčena jiná NA. NA se na začátku každé funkce zálohuje na vrchol zásobníku do tzv. RAP cookie a na konci je ověřováno, že se na vrcholu zásobníku nachází tatáž hodnota.

```
push %ebp        ; Uložení hodnoty registru EBP -- SFP (ukazatele na začátek nadřazeného AZ) -- na
 ↪  zásobník.
movl %esp,%ebp   ; Prohlášení adresy vrcholu zásobníku (uložené v ESP) za začátek aktuálního AZ
 ↪  (uložením do EBP).
subl 16,%esp     ; Vymezení prostoru pro lokální proměnné na zásobníku (zde 16 B).
```

Strojový kód x86 9.1: Standardní prolog funkce na architektuře x86-32

```
push %ebp
movl %esp,%ebp
subl 16,%esp
xor %r12,%ebp    ; Uložení RAP kanárka -- XOR(<náhodná-hodnota-v-R12>,EBP>) -- do registru EBP.
                 ; Registr EBP neobsahuje skutečnou adresu začátku aktuálního AZ, nýbrž její zobrazení
                  ↪  XOR(R12,EBP).
```

Strojový kód x86 9.2: Prolog funkce na architektuře x86-32 po aplikaci RAP

```
movl %ebp,%esp   ; Odstranění obsahu AZ změnou hodnoty registru ESP. Vrcholem zásobníku se stane
 ↪  počátek AZ.
popl %ebp        ; Ukazatel na začátek AZ se změní na svou původní hodnotu, která byla doteď uložena
 ↪  na zásobníku.
popl %eip        ; Vyzvednutí návratové adresy ze zásobníku a její uložení do registru EIP.
```

Strojový kód x86 9.3: Standardní epilog funkce na architektuře x86-32

```
xor %r12,%ebp     ; Rozšifrování původní NA hodnotou uloženou v R12 a její uložení do registru EBP.
cmp %ebp,24(%esp) ; Porovnání aktuální návratové adresy (uložené na adrese ESP-24) s původní adresou
 ↪  v uloženov v EBP.
jnz .error        ; Jsou-li NA rozdílné => skok na návěští .error.
movl %ebp,%esp
popl %ebp
popl %eip
.error:
ud2               ; Útok detekován => ukončení procesu.
```

Strojový kód x86 9.4: Epilog funkce na architektuře x86-32 po aplikaci RAP

RAP přepisuje epilogy všech funkcí tak, že před instrukcí `ret` (čili `popl eip`) jsou vždy umístěny instrukce kontrolující RAP cookie, což znepříjemňuje nalezení vhodných ROP gadgetů pro sestavení exploitu. Hodnota registru `%R12` je volena náhodně. Útočník tak nemá možnost změny návratové adresy funkce. Technika ROP **není odolná vůči ASLR**, nicméně k jejímu úspěšnému nasazení stačí přítomnost alespoň jedné dynamicky linkované knihovny, na které není aplikováno ASLR.

9.17 Řešení nedostatků v zabezpečení chroot prostředí

Účelem vylepšení Grsecurity týkajících se chroot prostředí jsou redukce možností neoprávněného opuštění chroot prostředí a omezení Inter-Process Communication (IPC) mezi v chroot prostředí uzavřenými procesy a procesy běžícími vně daného chroot prostředí. Toho Grsec dosahuje znepřístupněním vybraných systémových volání vězněným procesům. Většina vylepšení je zaměřena na situaci, kdy nepřátelský subjekt běžící v chroot prostředí disponuje superuživatelskou identitou, bez které, vzhledem ke své, přestože nedokonalé, izolaci, nepřestavuje hrozbu. Nedostatky zabezpečení chroot prostředí včetně jejich řešení poskytnutými Grsecem jsou popsány v tabulce 9.9.

Tabulka 9.9: Volby poskytnuté při konfiguraci jádra obohaceného o Grsecurity pro posílení bezpečnosti chroot prostředí

Volba	Dopad zvolení volby
`GRKERNSEC_CHROOT` Chroot jail restrictions	Zpřístupní volby GrSec týkající se chroot prostředí, které, budou-li zapnuty, učiní únik z něj obtížnější. Standardně je skutečnost, že se proces nachází v chroot prostředí, reflektována pouze ve vybraných funkcích jádra manipulujících s FS. Grsec modifikuje funkce, které skutečnost, že se subjekt je volající nachází v chroot prostředí a že by je mohl **zneužít pro interakci s objekty** vně toto chroot prostředí, neberou v úvahu.
`GRKERNSEC_CHROOT_MOUNT` Deny mounts	Subjektům v chroot prostředí nebude přístupné systémové volání `mount()`, respektive nebudou moci si připojit zařízení (FS) do prostoru svého chroot prostředí [Buc15]. Subjekt voláním `chroot()` změní své chroot prostředí na jím připojený FS. S nastavením volby lze manipulovat z US nastavováním sysctl proměnné `kernel.grsecurity.chroot_deny_mount`
`GRKERNSEC_CHROOT_DOUBLE` Deny double-chroots	Procesy běžící uvnitř chroot prostředí ztratí možnost provádět systémové volání `chroot()`. Bez aplikace této ochrany se mohou subjekty vymanit z chroot prostředí vytvořením nového chroot prostředí mimo to stávající [Luk13].
`GRKERNSEC_CHROOT_PIVOT` Deny pivot_root in chroot	Subjekty v chroot prostředí nemohou použít systémové volání `pivot_root()`, které mění kořenový adresář. S nastavením volby lze manipulovat z US nastavováním sysctl proměnné `kernel.grsecurity.chroot_deny_chroot`
↓	Pokračování tabulky na další straně.

Pokračování tabulky z předchozí strany. ↑

Volba	Dopad zvolení volby
GRKERNSEC_CHROOT_CHDIR Enforce chdir("/")	Volba vynutí přesun subjektu do kořene chroot prostředí ihned po provedení systémového volání `chroot()`. Zapnutí této volby znemožní opuštění chroot prostředí prováděné následujícím způsobem: 1. Subjekt v chroot prostředí získá superuživatelskou identitu. 2. Subjekt vytvoří adresář `mkdir <dir>`. 3. Subjekt se uzavře do nového chroot prostředí – `chroot <dir>`. 4. Subjekt tímto unikl z jeho předchozího chroot prostředí [Luk13]. S nastavením volby lze manipulovat z US nastavováním sysctl proměnné `kernel.grsecurity.chroot_enforce_chdir`
GRKERNSEC_CHROOT_CHMOD Deny (f)chmod +s	Subjekty nejsou schopny prostřednictvím příkazu `chmod` modifikovat přístupová oprávnění souborů v chroot prostředí za účelem aktivace SUID/SGID bitů na nich. S nastavením volby lze manipulovat z US nastavováním sysctl proměnné `kernel.grsecurity.chroot_deny_chmod`
GRKERNSEC_CHROOT_FCHDIR Deny fchdir out of chroot	Znemožní uvězněným subjektům volání `fchdir()` nad file deskriptorem ukazujícím do souboru ležícího mimo chroot prostředí [Luk13]. S nastavením volby lze manipulovat z US nastavováním sysctl proměnné `kernel.grsecurity.chroot_deny_fchdir`
GRKERNSEC_CHROOT_UNIX Deny access to abstract AF_UNIX sockets out of chroot	Subjekty v chroot prostředí se nebudou moci připojit k doménovým – `AF_UNIX` – socketům navázaným na subjekty běžící mimo chroot prostředí. S nastavením volby lze manipulovat z US nastavováním sysctl proměnné `kernel.grsecurity.chroot_deny_unix`
GRKERNSEC_CHROOT_FINDTASK Protect outside processes	Subjekty uvnitř chroot prostředí nemohou interagovat se subjekty mimo něj, a to ani vzájemným posíláním signálů. S nastavením volby lze manipulovat z US nastavováním sysctl proměnné `kernel.grsecurity.chroot_findtask`
GRKERNSEC_CHROOT_NICE Restrict priority changes	Subjekty v chroot prostředí nejsou oprávněny zvyšovat prioritu procesů běžících mimo chroot prostředí. S nastavením volby lze manipulovat z US nastavováním sysctl proměnné `kernel.grsecurity.chroot_restrict_nice`
GRKERNSEC_CHROOT_SYSCTL Deny sysctl writes	Odebere subjektům běžícím v chroot prostředí možnost modifikovat sysctl proměnná. Nebudou je moci modifikovat ani voláním `sysctl()`, ani zápisem do `proc` FS. S nastavením volby lze manipulovat z US nastavováním sysctl proměnné `kernel.grsecurity.chroot_deny_sysctl`

Konec tabulky.

9.18 Přidání podpory MAC/RBAC

Grsec přidává do jádra vlastní řešení RBAC, respektive MAC, které je postaveno na ACL v kontextu Grsec dále nazývaného Grsecurity ACL.

Grsecurity ACL jsou **procesně-orientovaná ACL**, která pracují s objekty, subjekty (viz kapitola 5.2) a rolemi.

9.18.1 Role

Role je **množina operací**, které smí subjekt provádět nad vybranými objekty, disponuje-li danou rolí.

Zavedením rolí se mění vlastník oprávnění, kterým již není subjekt nýbrž role. Role tak představují další vrstvu, která se nachází mezi přidělenými přístupovými oprávněními a subjekty zpřehledňující konfiguraci AC a umožňující specifikovat situace, ve kterých bude přístup subjektu povolen namísto povolení přístupu subjektu v jakýkoliv okamžik [Moš].

RBAC je o systém rolí obohacené MAC. RBAC práva používaná v GrSecurity jsou variantou MAC práv, jež pro odlišení jednotlivých nastavovaných subjektů využívají cestu k nim vedoucí v souborovém systému. Zatímco PaX má dopad na veškeré subjekty v systému, RBAC omezuje každý subjekt zvlášť prostřednictvím GrSecurity ACL.

Subjekt má vždy přiřazenu roli [Spe03]. Definice role má tvar uvedený v konfiguraci Grsec 9.1. Tato definice povinně

```
role <identifikátor-role> <volitelné-příznaky-role>
    [ role_transitions  <role1> [<role2> ...] ]
    [ role_allow_ip <ip-adresa>/<maska-sítě> ]
```

Konfigurace GrSec 9.1: Struktura definice role v Grsec

obsahuje

`<identifikátor-role>`,
: Role je identifikována řetězcem, jehož podoba se odvíjí od typu role. Možné identifikátory rolí jsou vyjmenovány v tabulce 9.10.

`<volitelné-příznaky-role>`
: Umístěním jednoho z příznaků uvedených v tabulce 9.10 mezi tyto příznaky lze specifikovat typ role. Příznaky uvedenými v tabulce 9.11 lze dále specifikovat vlastnosti role.

a může obsahovat též

`role_transitions <role1> [<role2> ...]`,
: Výčtem definovaná množina rolí, do kterých Grsec umožní přechod účastníkovi dané role.

`role_allow_ip <ip-adresa>/<maska-sítě>` **a**
: Určení IP adres uchazečů o danou roli, kterým bude žádost o ní akceptována.

definice subjektů.
: Subjekty, respektive absolutní cesty k nim, uvedené za definicí role mají příslušnou roli nastavenu jako výchozí.

Název role	Požadována autentizace heslem	Identifikátor role	Příznak	Charakteristika role
uživatelská	NE	system username	u	Role se vztahuje na procesy běžící pod určitou uživatelskou identitou.
skupinová	NE	system groupname	g	Role se vztahuje na procesy běžící pod určitou skupinovou identitou.
standardní	NE	řetězec `default`	G	Role se vztahuje na procesy, které nepatří do žádné uživatelské role, které nepatří do žádné skupinové role a které se do žádné role neautorizovaly. Role tohoto typu vždy v systému existuje a vždy je pouze jedna.
speciální	ANO	???	s	Role, na kterou se nevztahují vybrané restrikce.

Tabulka 9.10: Typy rolí rozpoznávané v AC Grsec RBAC

Příznak	Význam příznaku
N	Do role bude nutné se autentizovat, avšak bez zadávání hesla.
G	Oprávnění role budou rozšířena tak, aby bylo možné využít nástroje `gradm` za účelem autentizace do jiné role.
T	Na subjekty běžící v dané roli bude uplatňováno Trusted Path Execution (TPE).
l	Subjekty běžící v dané roli mají zapnutý learning mode.

Tabulka 9.11: Příznaky upravující chování role [Spe03]

9.18.1.1 Přechody mezi rolemi

Přiřazení role subjektu je prováděno následovně:

1. Subjektu je přiřazena uživatelská role, existuje-li. Pokud ne ⇒ 2.
2. Subjektu je přiřazena skupinová role, existuje-li. Pokud ne ⇒ 3.
3. Subjektu je přiřazena výchozí[8] role.

Subjekty mohou přecházet mezi rolemi, avšak pouze ve směrech

$$\text{(uživatelská|skupinová|defaultní) role} \rightarrow \text{speciální role}$$
$$\text{speciální role} \rightarrow \text{(uživatelská|skupinová|defaultní) role}$$

9.18.1.2 Manipulace s rolemi

Pro manipulaci s rolemi poskytuje Grsec nástroj `gradm`, který umožňuje mimo jiné přihlášení uživatele do role, `gradm -a <název-role>` přihlášení uživatele do role nezabezpečené heslem či `gradm -n <název-role>` nastavení hesla zabezpečujícího získání role. `gradm -P <název-role>`

9.18.2 Konfigurace GrSecurity ACL

GrSecurity ACL jsou konfigurovány z US nástrojem `gradm` [Spe03]. Ten je standardně přístupný pouze subjektům s privilegovanou identitou. Modifikace GrSecurity ACL jsou chráněny vlastní autentizací subjektu heslem.

9.18.3 Grsec politika

Grsec politika (GRP) je označení pro záznam v GrSecurity ACL, respektive pro definici přístupových oprávnění **právě jednoho** subjektu v systému.

GRP jsou zapisovány v podobě obyčejných textových souborů do adresáře `/etc/grsec/acl`. Jejich parsování neprovádí jádro, nýbrž nástroje `lex` a `yacc` běžící v User Space, jejichž výstup je předán jádru prostřednictvím zápisu do souborového systému `/proc`.

9.18.3.1 Vytvoření politiky

GRP lze vytvořit libovolnému subjektu či objektu, který ani nemusí existovat [Moš]. GRP

- vztažená k subjektu bude dále označována jako GRP-S a
- vztažená k objektu analogicky – GRP-O.

GRP je možné zkonstruovat

ručně
: Zapsání politiky ve tvaru vyobrazeném v Grsec konfiguraci 9.2.

nebo poloautomaticky.
: Využití existence learning mode:

  1. Systém je přepnut do learning mode.
  : Přepnutí do learning mode lze provést

    pro celý systém
    : Přepnutí zajistí příkaz `gradm -F -L <cesta-k-souboru-s-logy>`

    nebo pouze pro vybrané subjekty.
    : Přepnutí zajistí příkaz `gradm -L <cesta-k-souboru-s-logy>` v kombinaci s přidáním příznaku `l` ke všem subjektům, jejichž činnost má být zaznamenávána.

[8] Výchozí – řetězcem `default` identifikovaná – Grsec role vždy existuje.

Od okamžiku přepnutí do learning mode je započato zaznamenávání událostí.

2. Iniciuje se aktivita subjektu, pro který je vytvářena politika.

Subjekt by během learning mode měl provést ideálně veškeré činnosti, které při svých běžích bude v budoucnu potřebovat. Subjekt je v tuto chvíli spuštěn **pod superuživatelskou identitou**.

3. Deaktivuje se Learning mode a vytvoří nové politiky.

Deaktivaci learning mode a vytvoření nových politik pro sledované subjekty na základě v něm zachycených dat zajistí příkaz `gradm -F -L <cesta-k-souboru-s-logy> -O <cesta-k-souboru-s-novými-politikami>` respektive `gradm -L <cesta-k-souboru-s-logy> -O <cesta-k-souboru-s-novými-politikami>` Grsec vytváří politiku na základě předpokladu, že veškeré zaznamenané akce prováděné subjektem jsou oprávněné a potřebné pro jeho správné fungování.

5. Navržená politika je předložena člověku – administrátorovi – k validaci.

Pokud člověk **pojme podezření**, že zaznamenané aktivity subjektu jsou pro jeho běh nepotřebné, politiku typicky neschválí.

9.18.3.2 Struktura politiky

GRP-S je zapisována ve formátu zobrazeném v konfiguraci Grsec 9.2, ve kterém se mohou nacházet

```
<identifikátor-subjektu> <volitelné-příznaky-subjektu>
{
    <identifikátor-objektu-typu-soubor> <volitelné-příznaky-objektu-typu-soubor>

    [+|-] <identifikátor-capability>

    <identifikátor-systémového-zdroje> <soft-limit> <hard-limit>

    connect
    {
        <ip>/<netmask>:<low-port>-<high-port> <type> <protocol>
    }

    bind
    {
        <ip>/<netmask>:<low-port>-<high-port> <type> <protocol>
    }
}
```

Konfigurace GrSec 9.2: Struktura GRP-S [Spe03]. Červeně vyznačená část představuje umístění a syntaxi definice GRP-O

`<identifikátor-subjektu>`,

Absolutní adresa

adresáře

V případě adresáře se GRP vztahuje ke všem spustitelným souborům v něm obsaženým. Vůli GRP má šanci změnit pouze GRP, která specifikuje přesněji absolutní cestu k ovlivňovanému souboru.

či spustitelného souboru

Je-li identifikátorem subjektu v GRP přímo spustitelný soubor, má tato GRP rozhodující slovo v určování přístupových oprávnění daného subjektu.

v rámci FS.

`<volitelné-příznaky-subjektu>`,

Nejvýznamější příznaky, kterými lze modifikovat vlastnosti jednotlivých subjektů jsou uvedeny v tabulce 9.12.

`<identifikátor-objektu-typu-soubor>`,

Absolutní adresa souboru v rámci FS, k němuž je řízen přístup daného subjektu Grsec.

`<volitelné-příznaky-objektu-typu-soubor>`,

Definice GRP vztahující se k objektu – GRP-O – se uvádí vždy v rámci definice GRP-S.

Vybrané příznaky, kterými lze modifikovat vlastnosti jednotlivých objektuu jsou uvedeny v tabulce 9.13.

`<identifikátor-capability>`,

Pojmenování subjektu udělené/odebrané capability. O capabilities se zmiňuje kapitola 5.5.

V zápisu GRP lze uvést **virtuální** capabilitu `CAP_ALL`, jejíž přidělení/odebrání subjektu způsobí přidělení/odebrání všech capabilit.

`<identifikátor-systémového-zdroje> <soft-limit> <hard-limit>`,

Identifikátory systémových zdrojů jsou vyjmenovány na `http://en.wikibooks.org/wiki/Grsecurity/Appendix/System_Resources`. Pro příklad konfigurace Grsec

`RES_FSIZE 10K 20M`

zapříčiní, že jí stižený proces nebude schopen vytvořit na FS soubor větší nežli 20 MB.

`<ip>/<netmask>:<low-port>-<high-port> <type> <protocol>`,

Definice oprávnění subjektu

- navazovat komunikaci (`connect`) či
- naslouchat (`bind`)

na příslušných portech.

Příznak	Dopad uvedení příznaku na subjekt
`p`	Označí subjekt jako chráněný – tzn. že jím označený subjekt bude moci být ukončen pouze subjektem s příznakem `k`.
`k`	Subjekt jím označený získá pravomoc ukončovat chráněné subjekty.
`d`	Zabrání subjektu přístup k **adresáři** `/proc/<pid>/fd` **a** Adresář `fd` obsahuje subjektem používané file deskriptorem, což krom souborů mohou být například i sokety. **souboru** `/proc/<pid>/mem`. Informace o adresním prostoru procesu je pro útočníka velmi cenná. Tímto opatřením se procesu, a tím pádem i útočníkovi, zamezí v přístupu k jinak pro něj čitelné části souborového systému `/proc`.
`K`	Pokusí-li se subjekt o přístup k objektu, ke kterému nemá přístupová oprávnění, bude ukončen.
`C`	Pokusí-li se subjekt o přístup k objektu, ke kterému nemá přístupová oprávnění, bude ukončen nejen on, ale také všechny subjekty komunikující s toutéž IP.
`h`	Skryje subjekt před všemi ostatními subjekty, krom těch s příznakem `v`.
`v`	Subjekt jím označený uvidí též skryté subjekty.
`l`	Zapne learning mode pro daný subjekt.
`o`	Odebere subjektu případná přístupová oprávnění zděděná z méně specifických GRP-S. Všechna oprávnění subjektu jsou definována v maximálně specifickém GRP-S.
`X`	Zapne[1] randomizaci adresního prostoru (ASLR) subjektu typu RANDEXEC (viz kapitola 9.1.6).
`R`	Vypne[2] randomizaci adresního prostoru (ASLR) subjektu typu RANDMMAP (viz kapitola 9.1.5).
`A`	Znepřístupní sdílenou paměť subjektu všem subjektum, kteří nepředstavují instance téhož spustitelného souboru.
`P`	Vypne[3] vynucování nespustitelných stránek ve VAS subjektu PAGEEXEC (viz kapitola 9.2.2.1).

Tabulka 9.12: Vybrané příznaky akceptované v definici subjektu v GRP-S

9.18.4 Řízení změny identity subjektu

Grsec umožňuje specifikovat identity, do nichž může subjekt konvertovat [Spe03]. Formát zápisu pravidel omezujících změnu identity subjektu zachycuje konfigurace Grsec 9.3 a konkrétní příklad pak konfigurace Grsec 9.4.

```
subject <identifikátor-subjektu>
    user_transition_allow <sys-username> [<sys-username> ...]
    group_transition_allow <sys-groupname> [<sys-groupname> ...]
```

Konfigurace GrSec 9.3: Syntaxe zápisu pravidla omezujícího možnost změny identity subjektu

9.18.4.1 Příklad

V Grsec konfiguraci 9.5 je uveden příklad jedné GRP, která

se vztahuje výhradně k programu `date`,

Řetězec `/bin/date` vztahuje celou GRP na procesy vykonávající obsah souboru `/bin/date`.

[1] ASLR v podobě RANDEXEC je nutné aktivovat u každého subjektu zvlášť.
[2] ASLR v podobě RANDMMAP je aktivní pro všechny subjekty automaticky.

Příznak	Dopad uvedení příznaku na objekt
r	objekt lze číst
w	Do objektu lze zapisovat.
x	Obsah objektu lze vykonávat.
c	objekt lze vytvořit.
d	objekt lze odstranit.
s	Zamítnutí přístupu k objektu **nebude** logováno.
i	Subjekt vzniklý spuštěním objektu subjektem, v jehož GRP-S je obsažený GRP-O s daným příznakem, zdědí GRP-S svého rodiče.

Tabulka 9.13: Vybrané příznaky akceptované v GRP-O [Spe03]

```
subject /bin/su
    user_transition_allow root
    group_transition_allow root
```

Konfigurace GrSec 9.4: Příklad pravidla omezujícího možnost změny identity subjektu `/bin/su` pouze na uživatelskou (a skupinovou) identitu `root`

```
/bin/date oA
{
    /usr/share r
    /etc r
    /usr/lib rx
    /var/run/date.pid w
    ...
}
```

Konfigurace GrSec 9.5: Příklad GRP

odebírá zděděná přístupová oprávnění,
Volba `o` odebere přístupová oprávnění, zděděná z nadřazených adresářů.
uděluje přístupová oprávnění k vyjmenovaným souborům.
Proces vykonávající soubor `/bin/date` smí

- číst obsah souboru `/usr/share`,
- číst obsah souboru `/etc`,
- číst a spouštět obsah souboru `/usr/lib` a
- zapisovat do souboru `/usr/run/date.pid`.

9.18.4.2 Podmínění platnosti

Platnost jednotlivých GRP-S lze podmínit identitou rodičovského subjektu. Syntaxe podmínky je uvedena v konfiguraci Grsec 9.6. Toto podmínění platnosti lze vnořovat (viz konfigurace 9.7). Vnořené subjekty dědí GRP-S svých rodičů

```
subject <identifikátor-rodičovského-subjektu>:<identifikátor-subjektu> <příznaky>
{
    ...
}
```

Konfigurace GrSec 9.6: Syntaxe podmínky omezující platnost GRP pouze na případy, kdy je rodičem subjektu určitý konkrétní subjekt

```
subject <id-subjektu>:<id-subjektu>:<id-subjektu>:<id-subjektu> <příznaky>
{
    ...
}
```

Konfigurace GrSec 9.7: Syntaxe podmínky omezující platnost GRP rekurzivně vnořená

– subjektuu, které je spustily.

9.18.5 Princip Grsec MAC

Princip AC bude vysvětlen na příkladu spuštění programu `/bin/date`.

1. Subjekt s identitou `U` iniciuje spuštění `/bin/date`.
2. Existuje GRP-S pro `/bin/date`? Ano. ⇒ 5
3. Existuje GRP-S pro `/bin`? Ano. ⇒ 5
4. Existuje[1] GRP-S pro `/`.
5. Pro AC – hledání GRP-O – daného subjektu použij nalezenou – nejvíce specifickou – GRP-S.
6. Právě spuštěný subjekt chce zapsat své PID do `/var/run/date.pid`.
7. Existuje GRP-O pro `/var/run/date.pid`? Ano. ⇒ 11
8. Existuje GRP-O pro `/var/run`? Ano. ⇒ 11
9. Existuje GRP-O pro `/var`? Ano. ⇒ 11
10. Existuje[1] GRP-O pro `/`.
11. Dle nalezené – nejvíce specifické – GRP-O rozhodni o udělení oprávnění k přístupu k objektu.
12. Umožňuje GRP-O pro `/var/run/date.pid` zápis? Ne. ⇒ 14
13. Přístup subjektu k objektu povolen MAC.
14. Přístup subjektu k objektu odepřen MAC.

9.18.5.1 Nastavení GrSecurity ACL

Konfiguraci GrSecurity ACL provádějící osoba může využít služeb nástroje `gradm` – nástroje jímž lze

- prohlížet logy Grsec,
- zapínat/vypínat learning mode či
- se přihlašovat do speciálních rolí.

[1] GRP-S i GRP-O pro / existují vždy. Subjekt definovaný cestou / představuje v Grsec tzv. standardní subjekt. **Musí být definován** pro každou roli.

9.19 Zvýšení bezpečnosti síťového subsystému

Volby aktivující úpravy jádra Grsecurity týkající se síťového zásobníku jsou uvedeny v tabulce 9.14. Při konfiguraci jádra jsou umístěny v sekci „Network protections“.

Tabulka 9.14: Volby Grsec týkající se zabezpečení síťového zásobníku

Volba	Dopad zvolení volby
GRKERNSEC_BLACKHOLE TCP/UDP blackhole and LAST_ACK DoS prevention	Zamezení odpovědí z portů, s nimiž není asociován žádný proces Zapnutí volby GRKERNSEC_BLACKHOLE **zamezí** praktikování následujících, standardních vzorců chování protokolového zásobníku na všech síťových rozhraních s výjimkou rozhraní loopback: • Počítač odešle TCP paket s nastaveným příznakem RST jako odpověď na paket směřující na port, s nímž není asociován žádný proces [RFC793]. • Počítač odešle ICMP zprávu Destination unreachable jako odpověď na paket směřující na port, s nímž není asociován žádný proces [RFC792]. V daných situacích tak nebude počítač činit nic. Důsledkem toho je • zvýšení odolnosti počítače proti DoS útokům, • snížení viditelnosti počítače pro síťové skenery a • menší spotřeba zdrojů v již zmiňovaných situacích. Sysctl proměnnou, kterou lze tuto volbu nastavovat za běhu systému, je kernel.grsecurity.ip_blackhole .
GRKERNSEC_NO_SIMULT_CONNECT Disable TCP Simultaneous Connect	Nedodržení TCP v zájmu bezpečnosti Zapnutí této volby odstraní zranitelnost plynoucí z důsledného dodržování definice TCP popsanou v kapitole 8.10. K odstranění této zranitelnosti provede GrSec jedinou úpravu ve funkci tcp_rcv_synsent_state_process().
GRKERNSEC_SOCKET Socket restrictions	Omezení přístupu procesů k socketům Zapnutí této volby zpřístupní volby týkající se omezení přístupu k socketům. Zapnutí některé z nich omezuje možnost procesů manipulovat se sockety (viz tabulka 9.15), jejichž vlastníky jsou uživatelé náležící do skupin specifikovaných GID. Pro deklaraci GID skupiny, na kterou se omezení dané volbou bude vázat, existuje vždy volba téhož názvu obohaceného o příponu _GID. Omezení mohou být následující: • Procesu není dovoleno připojit se k serverům běžícím na tomtéž počítači. ⇒ Proces nemůže být klientem v rámci počítače. • Procesu není dovoleno naslouchat na jakémkoliv portu počítače. ⇒ Proces nemůže být serverem. Uživatelé, jimiž jsou spouštěny v testech zúčastněné programy nejsou členy skupin, členství v nichž GrSec trestá.
GRKERNSEC_SOCKET_ALL Deny any sockets to group	Zapnutí této volby a následná specifikace GID skupiny ve volbě GRKERNSEC_SOCKET_ALL_GID má stejný dopad, jako současné zapnutí voleb • GRKERNSEC_SOCKET_CLIENT (viz volba „Deny client sockets to group“) a • GRKERNSEC_SOCKET_SERVER (viz volba „Deny server sockets to group“). pro skupinu s daným GID zároveň. Sysctl proměnné s touto volbou spjaté jsou kernel.grsecurity.socket_all a kernel.grsecurity.socket_all_gid .

↓ Pokračování tabulky na další straně.

Pokračování tabulky z předchozí strany. ↑

Volba	Dopad zvolení volby
GRKERNSEC_SOCKET_CLIENT Deny client sockets to group	Zapnutí této volby zajistí, že kterýkoliv uživatel náležící do skupiny s GID uvedeným ve volbě GRKERNSEC_SOCKET_CLIENT_GID, **nebude** mít možnost se připojit k serverům běžícím na tomtéž počítači. Ve volbě obohacené suffixem _GID se specifikuje GID skupiny, které se omezení týká. Sysctl proměnné s touto volbou spjaté jsou kernel.grsecurity.socket_client a kernel.grsecurity.socket_client_gid .
GRKERNSEC_SOCKET_SERVER Deny server sockets to group	Zapnutí této volby zajistí, že kterýkoliv uživatel náležící do skupiny s GID uvedeným ve volbě GRKERNSEC_SOCKET_SERVER_GID, **nebude** mít možnost být serverem – naslouchat na libovolném portu počítače. Sysctl proměnné s touto volbou spjaté jsou kernel.grsecurity.socket_server a kernel.grsecurity.socket_server_gid .

Konec tabulky.

měřená konfigurace	možnost být lokálním klientem	možnost být serverem
GRKERNSEC_SOCKET_ALL	✖	✖
GRKERNSEC_SOCKET_CLIENT	✖	neřeší
GRKERNSEC_SOCKET_SERVER	neřeší	✖

Tabulka 9.15: Možnosti omezení nakládání se sockety poskytované patchem GrSec

9.20 Znemožnění přístupu k /dev/kmsg

Aktivace volby GRKERNSEC_DMESG z kategorie „Executable options“ způsobí, že neprivilegovanému uživatel budou viditelné pouze poslední 4 kB z /dev/kmsg. Příkaz dmesg bude tak moci používat i nadále, avšak pouze pro nejmladší historii událostí. Sysctl proměnná modifikující tuto volbu za běhu systému se jmenuje kernel.grsecurity.dmesg.

Podobnou funkcionalitu poskytuje i systémová proměnná kernel.dmesg_restrict zavedená roku 2010 do jádra (od verze jádra 2.6.37) [Ros10]. Tato volba znepřístupní /dev/kmsg neprivilegovaným uživatelům celý za základě požadavku na capabilitu CAP_SYS_ADMIN.

9.21 Řízení přístupu k souborovému systému /proc

Grsec přináší též ochranu souborového systému /proc. Jednotlivé volby a jejich dopad je popsán v tabulce 9.16.

Tabulka 9.16: Volby Grsec zabezpečující souborový systém /proc

Volba	Dopad zvolení volby
GRKERNSEC_PROC Proc restrictions	Zpřístupní zbylé volby omezující přístupová práva k virtuálnímu souborovému systému mapovanému do adresáře /proc. Dále omezí pohled každého jednotlivého uživatele na spuštěné procesy. Uživatel uvidí vždy pouze své procesy – procesy, jichž je (pra)rodičem. Toto omezení výrazně znesnadní situaci případného útočníka.

↓ Pokračování tabulky na další straně.

Pokračování tabulky z předchozí strany. ↑

Volba	Dopad zvolení volby
`GRKERNSEC_PROC_USER` Restrict `/proc` to user only	Aktivace této volby zapříčiní stav, kdy **neprivilegovaní uživatelé uvidí výhradně své vlastní procesy**.
`GRKERNSEC_PROC_USERGROUP` Allow special group	Uživatelé, kteří nejsou členy skupiny, jejíž GID je rovno hodnotě volby GRKERNSEC_PROC_GIDuvidí pouze vlastní procesy.
`GRKERNSEC_PROC_GID` GID exempted from `/proc` restrictions	Umožní specifikovat GID skupiny, na jejíž členy se omezení dané `GRKERNSEC_PROC_USER` nebude vztahovat.
`GRKERNSEC_PROC_ADD` Additional restrictions	Omezí přístup neprivilegovaných uživatelů k souboru `/proc/slabinfo`, čímž tito uživatelé přijdou o možnost sledovat obsah cache jádra umístěné v Kernel Space, do které jsou ukládány nejčastěji přistupované inode, dentry a mnohé další objekty. Informace o v ní obsažených datech se považují za zneužitelné při exploitování řady zranitelností systému.

Konec tabulky.

9.22 Další drobná vylepšení bezpečnosti

Grsec vylepšuje zabezpečení v mnohých dalších oblastech. Volby, které byly vyhodnoceny jako „dostatečně zajímavé“ pro prostředí CIV ZČU jsou popsány v tabulce 9.17.

Tabulka 9.17: Dalčí drobná vylepšení bezpečnosti v rámci Grsec

Volba	Dopad zvolení volby
`GRKERNSEC_LINK` Linking restrictions	V adresářích s nastaveným sticky bitem (typicky `/tmp`) znemožní následování symbolických odkazů odkazujících na soubor vlastněný jiným uživatelem. To je výhodné pro zamezení zneužití chyb souběhu (race condition) spojených s práci se soubory v programech. Typickým příkladem podobné chyby je situace, kdy SUID spustitelný soubor používá pro zápis dočasný soubor v adresáři `/tmp`, jehož název je ovšem předvídatelný a útočník předtím vytvoří v `/tmp` symbolický odkaz se stejným jménem, jako bude mít tento dočasný soubor. Útočníkem vytvořený odkaz pak ukazuje na zcela jiný soubor, do kterého by za normálních okolností neměl daný uživatel právo zapisovat. Nicméně takto děravý program zápis provede a tím umožní útočníkovi soubor přepsat. Tuto možnost útoku zapnutí volby `GRKERNSEC_LINK` znemožní. S nastavením této volby lze manipulovat prostřednictvím sysctl proměnné `kernel.grsecurity.linking_restrictions` za běhu Linuxu.
`GRKERNSEC_FIFO` FIFO restrictions	Zakáže zápis do rour v adresářích s nastaveným sticky bitem, které uživatel nevlastní (typicky `/tmp`), není-li vlastník roury též vlastníkem adresáře, v němž roura leží. S nastavením volby lze manipulovat z US nastavováním sysctl proměnné `kernel.grsecurity.fifo_restrictions`.
`GRKERNSEC_PROC_MEMMAP` Harden ASLR against information leaks and entropy reduction	Zvyšuje bezpečnost randomizace adresního prostoru tím, že rozložení AS není možné zjistit výpisem speciálního souboru `/proc/<pid>/maps`.
`GRKERNSEC_HIDESYM` Hide kernel symbols	Zajistí, že aplikace bez sys. práva (capability) CAP_SYS_MODULE (viz kapitola 5.5) nebude mít možnost zjistit adresy jednotlivých funkcí v jádře, což může znesnadnit zneužití jaderných chyb.
`GRKERNSEC_ACL_HIDEKERN` Hide kernel processes	Při aktivovaném GrSec ACL systému dojde k zatajení existence procesů jádra uživatelským procesům.

Konec tabulky.

10. Srovnání úrovně zabezpečení GNU Linux

Kapitola 8 seznámila čtenáře s vybranými hrozbami. Kapitola 9 se následně pokusila vysvětlit možné techniky obrany vůči těmto hrozbám, které přináší patch Grsec/PaX, který byl zvolen k podrobnějšímu průzkumu v kapitole 7. V tabulce 10.1 je uveden dopad aplikace

Nasazení PaXu v částečné konfiguraci (např. ASLR bez NOEXEC) není právě výhodné, neboť eliminací pouze některých zranitelností hrozbě převzetí kontroly nad procesem obecně nelze předejít. Ochrany NOEXEC a ASLR je proto nanejvýš vhodné nasazovat zároveň, neboť samostatně jsou „obejitelné“.

Tabulka 10.1: Srovnání zabezpečení GNU Linux s a bez aplikovaného patche Grsec/PaX

Hrozba	Standardní zabezpečení systému (bez úprav jádra)	Nadstandardní zabezpečení systému (v podání Grsec)	Alternativní nadstandardní řešení
8.1.1 – vytvoření nového spustitelného mapování	**Systém chráněn.** Před danou hrozbou dokáží efektivně bránit i obyčejné DAC – standardní unixová práva (viz 5.3.3) či POSIX ACL (viz 5.3.4) dohlížející na přístupy k objektům.	**Systém chráněn.** Ochranu v tomto směru vylepšuje či lépe řečeno „činí pohodlnější pro její správu“ RBAC, který je součástí Grsec.	SELinux (viz kapitola 6.5), AppArmor (viz kapitola 6.4)
8.1.2 – shellcode injection na architektuře x86-64	**Systém částečně chráněn.** x86-64-kompatibilní procesory hardwarovou podporou NX bitu v Memory Management Unit disponují. Jádro 3.16 např. ochranou proti spouštění vybaví pouze mapování userspace-stack a haldu ponechá nadále spustitelnou. Útočníkovi navíc stále zbývá možnost jakoukoliv paměťovou stránku učinit spustitelnou, pokud se mu podaří zavolat `mprotect()`. V nových verzích jádra	**Systém chráněn.** Hrozbám z této kategorie efektivně brání technika NOEXEC, kdy ač se útočníkovi podaří neoprávněně přepsat paměťový prostor (typicky je příčinou logická chyba samotného programu), tak se mu jej díky NOEXEC a také MPROTECT nepodaří spustit.	
8.1.2 – shellcode injection na architektuře x86-32	**Systém nechráněn.** x86 procesory bez podpory 64 bitových instrukcí či lépe řečeno jejich MMU nedisponují podporou NX bitu. Obsah všech mapování lze spustit.	**Systém chráněn.** Grsec přináší podporu nespustitelných stránek na x86-32 architekturu a to softwarovými emulacemi využívajícími existence mechanismů, jejichž původní účel byl původně jiný – PAGEEXEC a SEGMEXEC.	Exec Shield
8.2 – vykonávání existujícího kódu mimo původní programové pořadí (útoky typu return-to-lib)	**Systém částečně chráněn.** Od Linuxu verze 2.6.12 jsou standardně znáhodňována umístění anonymních mapování – zásobníku a haldy.	**Systém chráněn.** Proti hrozbám této kategorie staví PaX techniku ASLR. Změna programového pořadí předpokládá změnu adresy skoku útočníkem. Obrana spočívá v utajení informace o uspořádání AS konkrétního procesu.	ASLR-NG

↓ Pokračování tabulky na další straně.

Pokračování tabulky z předchozí strany. ↑

Hrozba	Standardní zabezpečení systému (bez úprav jádra)	Nadstandardní zabezpečení systému (v podání Grsec)	Alternativní nadstandardní řešení
8.3 – Vykonání existujícího kódu v původním programovém pořadí s původními daty	**Systém nechráněn.** Ochranou proti hrozbám tohoto typu je nepoužívat programy této kategorie.	**Systém nechráněn.**	
8.6 – únik informace z kernelspace-stack	**Systém nechráněn.**	**Systém chráněn.** Grsec únik informace tímto způsobem eliminuje, a to volbou `PAX_MEMORY_STACKLEAK` (viz kapitola 9.14).	
8.7 – neoprávněné opuštění chrootu	**Systém nechráněn.**	**Systém chráněn.** Po aktivaci voleb uvedených v kapitole 9.17 systém hrozbu eliminuje.	
8.8 – ROP (rozdělení shellcode do mnoha malých úseků)	**Systém nechráněn.** Hrozba obchází na 64 bitových systémech aktivní ochranu userspace-stack před vykonáváním jeho obsahu – NX.	**Systém chráněn.** Grsec přichází s technikou RAP (viz kapitola 9.16, která zálohou návratové adresy útoku ROP čelí.	
8.9.2 – ret2usr (vykonávání kódu uloženého v User Space v režimu CPL 0)	**Systém chráněn.** Ovšem pouze za předpokladu, že běží na procesorech Intel Haswell či novějších – viz technika SMAP/SMEP v kapitole 9.5.3 a zároveň používá jádro 3.7 a novější, neb SMAP/SMEP by bez softwarové podpory nefungovalo.	**Systém chráněn.** Grsec nabízí proti ret2usr techniky UDEREF (viz kapitola 9.5.2) a KERNEXEC (viz kapitola 9.5.1. Obě se různými způsoby snaží oddělit jinak souvislý User Space a Kernel Space. Na x86-32 k tomu používají Linuxem jinak zcela opomíjenou segmentaci. Na x86-64 se již jejich přístup liší.	kGuard [KPK12]
8.9.2.1 – Return to direct mapped memory neboli ret2usr využívající přímé mapování	**Systém nechráněn.**	**Systém nechráněn.** Útok je postaven na přístupu přes přímé mapování, s jehož využitím ani jedna ze zmíněných zabezpečovací technik nepočítala.	9.5.4 – XPFO
8.10.1 – DoS díky důsledné implementaci TCP protokolu	**Systém nechráněn.** Minimálně vanilla jádro ctí definici TCP – RFC 793 [RFC793] – a ponechává i nadále schopnost TCP souběžně navazovat spojení aktivovanou.	**Systém chráněn.** Volba `GRKERNSEC_NO_SIMULT_CONNECT` zakáže podporu souběžného spojení.	není známo

Konec tabulky.

11. Měřící/měřené prostředí

Pro měření jsou k dispozici dva fyzické počítače – xenmv.civ.zcu.cz (xenmv) a metalist.civ.zcu.cz (metalist) – ležící ve stejné síti a komunikující přes neznámý počet přepínačů. Počítače tak nesdílí sice kolizní doménu, sdílí však doménu broadcastovou. Na metalist nedisponuje testující osoba možností výměny jádra, které tam je ve verzi $3.16.0$ bez patche GrSec. Xenmv možnost výměny jádra dává. Testy proběhnou na jádře $4.8.16$ při různých jeho konfiguracích. Hardwarové parametry počítačů jsou uvedeny v tabulce 11.1. Na obrázku 11.1 je znázorněn vztah jednotlivých prvků účastnících se měření. Infrastruktura schovaná pod obláčkem svými parametry vysoce převyšuje na obrázku 11.1 vynesené spoje a nehrozí, že by mohla být úzkým hrdlem testu.

Nechť jakýkoliv další výskyt slova *prostředí* označuje právě nyní popsané měřené/měřící prostředí, není-li v bezprostřední blízkosti jeho výskytu uvedeno jinak.

jméno počítače	**metalist**	**xenmv**
jméno multiprocesoru	Xeon	Xeon E5-2620 v2 [IntelArk]
počet procesorů	4	6
počet vláken vykonavatelných zároveň	4	12[1]
základní frekvence každého z procesorů	3,2 GHz	2,1 GHz
velikost hlavní paměti	8,195148 GB	16,382688 GB

Tabulka 11.1: Hardwarové parametry počítačů, které se zúčastní testování

Obrázek 11.1: Vizualizace měřícího/měřeného prostředí

11.1 Nejslabší článek spoje xenmv ↔ metalist

Několik testů uvedených v kapitole 13 zjišťuje právě propustnost spoje xenmv ↔ metalist. Jsou to měření v kapitolách 13.4.2, 13.5.5 a 13.5.6. Žádný z nich nenaměřil rychlost vyšší nežli 0,96 Gb/s čili 0,894 Gib/s. Úzkým hrdlem je nejpomalejší prvek na spoji. Příkaz 11.1 usvědčuje ethernetový adaptér na počítači xenmv z toho, že je jedním z množiny nejpomalejších prvků na spoji. **Maximální teoretická propustnost spoje činí 1 Gb/s respektive 0,931 Gib/s.**

```
root@xenmv2:~# ethtool eno1 | grep Speed
        Speed: 1000Mb/s
```

Příkaz/y 11.1: Rychlost síťového rozhraní na počítači xenmv

11.2 Redukce počtu procesorových jader

Autor dokumentu se po otestování jader odpovídajících konfiguracím, jejichž nastavení je uvedeno v tabulce 11.2, střetl s výsledky, které nelogicky znevýhodňovaly mnohé konfigurace, modifikující třeba jen síťový zásobník, ve výkonnostních testech škálujících přes počet do testu zapojených procesů. Vedoucí – Michal Švamberg – následně odhalil chybu, která byla příčinou: Za výkonnostním propadem stálo nastavení volby jádra `CONFIG_NR_CPUS`. U postižených konfigurací byla tato volba nastavena na hodnotu 8, zatímco ostatní konfigurace ji měly nastavenu na hodnotu 512. Volba svou hodnotou omezuje počet jader, které bude Linux s ní zkompilovaný schopen využít. Jelikož procesor na testovaném počítači (xenmv) disponoval 12 jádry, byly chybou stižené konfigurace handicapovány a to teoreticky až o ⅓ procesorového výkonu. Hodnotu 8 do volby `CONFIG_NR_CPUS` vpašovalo vypnutí volby `CONFIG_64BIT` během konstruování konfigurací pro kompilace jader. Vypnutí této volby bylo vynuceno sestavováním konfigurací noexec-pageexec a noexec-segmexec zachycujících volby jádra, které představují emulaci NX bitu. Tato emulace je dostupná pouze pro jádra běžící v režimu x86-32.

[2] Procesor Xeon E5-2620 v2 disponuje podporou technologie SMT a je tak schopen na každém svém jádru vykonávat dvě vlákna zároveň.

Jelikož autor tohoto dokumentu není schopen zkompilovat x86-32 jádro podporující více nežli 8 procesorových jader, vyvstal problém porovnání jednotlivých konfigurací. Proto je počet podporovaných jader – volba `CONFIG_NR_CPUS` – nastavena pro všechny konfigurace na hodnotu 8. Tímto se sice poněkud degraduje procesor testovaného stroje Intel E5-2620 v2, avšak jelikož pro všechny konfigurace stejně, nemá to na relativní srovnání jejich výkonnosti vliv.

11.3 Konfigurace měřeného prostředí

Konfigurace měřeného prostředí je definovaný stav testovaného prostředí (viz kapitola 11), do něhož bylo toto prostředí vmanipulováno předchozími jeho úpravami. Konfigurace měřeného prostředí se liší buďto jádrem použitým na počítači xenmv či nastavením sysctl proměnných (viz kapitola 11.4.2.1). Seznam konfigurací a výčet úprav v nich provedených je uveden v tabulce 11.2.

11.4 Příprava prostředí

Aby bylo co testovat, bude prostředí převáděno mezi různými stavy vyjmenovanými v tabulce 11.2. Převod prostředí do konkrétního stavu obnáší konkrétní požadavky. Určitý stav vyžaduje typicky

- **určitou verzi jádra**,

  Vyžaduje-li stav určitou verzi jádra, je zapotřebí tuto verzi jádra nejprve vytvořit (viz kapitola 11.4.1.2) a následně zvolit toto jádro při spuštění systému.
- **určité nastavení sysctl proměnných** a

  Mnohé volby, které jsou součástí stavu jsou nastavovány až dodatečně po nabootování systému. Bylo by sice možné pro každý stav zkompilovat samostatné jádro, avšak to by bylo neúnosně časově náročné. Proto se některé volby nastavují až příkazem `sysctl` nad jádrem, které je společné pro několik stavů.
- **případné další nastavení**.

  Je možné, že stav vyžaduje další modifikace a nastavení. Například běh testovacích skriptů pod privilegovanou identitou. O případných dalších nastaveních pro daný stav by měla v následujícím textu existovat zmínka.

Obecně lze úpravy, které jsou vyžadovány pro nastavení prostředí do stavu, rozdělit do dvou kategorií, které následují:

11.4.1 Úpravy prováděné před spuštěním měřícího/měřeného systému

11.4.1.1 Patch jádra

Zatímco SELinux a AppArmor jsou již ve vanilla jádře obsaženy, GrSec je nutné si do jádra dodatečně přidat, čehož lze v kořenovém adresáři se zdrojovými soubor jádra dosáhnout příkazem `patch`.

```
patch -p1 < soubor
```

C kód 11.1: Aplikace patche na jádro

11.4.1.2 Konfigurace jádra

Volby stanovené v rámci interakce uživatele s procesem spuštěným příkazem `make menuconfig` mohou být uloženy do souboru. Vzniknuvší konfigurační soubor má podobu seznamu voleb. Byla-li volba při konfiguraci zvolena, má v konfiguračním souboru vždy přiřazenu hodnotu. V opačném případě je zakomentována a doplněna dovětkem „is not set“.

```
# <KOMENTÁŘ>
<POJMENOVANI-VOLBY>=<HODNOTA-VOLBY>
# <POJMENOVANI-VOLBY> is not set
```

Konfigurace GrSec 11.1: Možné položky konfiguračního souboru vzešlého z `make menuconfig`

Veškeré volby mají prefix `CONFIG_`. Má-li uživatel konfigurační soubor k dispozici, nic mu nebrání měnit hodnoty přímo v něm. Nezakomentované volby jsou při kompilaci jádra nadefinovány jako makra preprocesoru

[3] Volba v dané konfiguraci takto nastavena povinně.

Pojmenování volby jádra	Jednotlivé konfigurace																
	original	autoconf_performance	autoconf_security	noexec	noexec_pageexec	noexec_segmexec	sanitize	stackleak	structleak	uderef	rap	aslr_basic	aslr_randmmap	rbac	blackhole	no_simult_connect	socket_limit
64BIT	✗	✓	✓	✓	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
GRKERNSEC	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
GRKERNSEC_CONFIG_AUTO	✗	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
GRKERNSEC_CONFIG_CUSTOM	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
GRKERNSEC_CONFIG_SERVER	✗	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
GRKERNSEC_CONFIG_PRIORITY_PERF	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
GRKERNSEC_CONFIG_PRIORITY_SECURITY	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
PAX	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	?	✓	✓	✗	✗	✗	✗
PAX_NOEXEC	✗	✓	✓	✓	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
PAX_PAGEEXEC	✗	✓	✓	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
PAX_SEGMEXEC	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
PAX_EMUTRAMP	✗	✓	✓	✗	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
PAX_MPROTECT	✗	✓	✓	✗	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
PAX_ASLR	✗	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓	✗	✗	✗	✗
PAX_RANDKSTACK	✗	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
PAX_RANDUSTACK	✗	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
PAX_RANDMMAP	✗	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗
GRKERNSEC_PROC_MEMMAP	✗	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗
GRKERNSEC_NO_RBAC	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✗	✗	✗
GRKERNSEC_BLACKHOLE	✗	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗
GRKERNSEC_NO_SIMULT_CONNECT	✗	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗
GRKERNSEC_SOCKET	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓
GRKERNSEC_SOCKET_ALL	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓
GRKERNSEC_SYSCTL	✗	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
GRKERNSEC_DMESG	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
PAX_MEMORY_SANITIZE	✗	✗	✓	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
PAX_MEMORY_STACKLEAK	✗	✗	✓	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗
PAX_MEMORY_STRUCTLEAK	✗	✗	✓	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗
PAX_MEMORY_UDEREF	✗	✗	✓	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗
PAX_RAP	✗	✓	✓	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗

Tabulka 11.2: Testované konfigurace systému čili volby jádra aktivované běžícího na stroji xenmv v době provádění testů

```
#define <POJMENOVANI-VOLBY> <HODNOTA-VOLBY>
```

C kód 11.2: Podoba volby při kompilaci

Existence (případně konkrétní hodnota) voleb následně při kompilaci podmiňuje překlad úseků v kódu jádra. Podmíněně překládaný úsek kódu obsahuje nezřídka jen několik málo příkazů.

```
if (sk->sk_state == TCP_TIME_WAIT) {
#ifdef CONFIG_GRKERNSEC_BLACKHOLE
ret = 2;
#endif
goto do_time_wait;
}
```

C kód 11.3: Konfigurační volby ovlivňují překlad prostřednictvím mechanismu podmíněného překladu

11.4.2 Úpravy prováděné po spuštění měřícího/měřeného systému

11.4.2.1 Konfigurace jádra za běhu

Zapnutí této volby `GRKERNSEC_SYSCTL` umožní změnu **vybraných** voleb GrSec za běhu (viz 11.4.2.1) tzn. bez nutnosti jádro znovu kompilovat příkazem `sysctl`. Možnost ovládat volby GrSec příkazem `sysctl` je pohodlná, ale zároveň představuje zranitelnost, neboť dostane-li se znepřátelený subjekt k privilegované identitě, potom si příkazem `sysctl` může deaktivovat mnohé z Grsec ochran. Proto je možné nastavení prováděné příkazem `sysctl` uzamknout – prohlásit za konečné – nastavením sysctl proměnné

```
kernel.grsecurity.grsec_lock
```

na hodnotu 1. **Zapnutí volby** `GRKERNSEC_SYSCTL` **způsobí vypnutí všech voleb GrSec** aktivovaných při konfiguraci jádra a zároveň přístupných aktivaci skrz příkaz `sysctl`. Zapnutí těchto voleb bude možné provést dodatečně:

V případě, že byla nastavena volba `GRKERNSEC_SYSCTL`, je možné měnit nastavení voleb GrSec, které to umožňují, za běhu. Změny se provádějí

- buďto zápisem do `/proc` (volby GrSec jsou umístěny v `/proc/sys/kernel/grsecurity`)
- nebo příkazem `sysctl` (viz posloupnost příkazů 11.1).

```
› sysctl -n <proměnná>             # čtení aktuální hodnoty proměnné
› sysctl -w <proměnná>=<hodnota>   # zápis hodnoty proměnné
```

Skript 11.1: Manipulace s konfigurací jádra za běhu

Pro všechny volby GrSec, které je možné měnit za běhu platí:

- Jejich ekvivalent pro příkaz `sysctl` začíná vždy prefixem „`kernel.grsecurity.`“.
- Pro jejich modifikaci je nutné, aby volba `kernel.grsecurity.grsec_lock` byla nastavena na 0. Změní-li hodnotu na !0, veškeré další změny GrSec voleb příkazem `sysctl` nebudou možné. Čili proměnná
  `kernel.grsecurity.grsec_lock`
  nevratně uzamyká možnost modifikace.

11.4.2.2 Nastavení voleb pro PaX

Významná část voleb PaX vyžaduje speciální softwarové vybavení pro využití poskytované funkcionality. Interagovat s PaX umožňuje nástroj `paxctl` či `chpax`. Nástroj je dostupný na adrese `http://www.grsecurity.net/download.php`. Závisí na zvoleném způsobu uložení příznaků pro PaX.

Nástroj `paxctl` slouží pro nastavení/odnastavení PaX konkrétního PaX příznaku pro konkrétní spustitelný soubor [AddUti]. Možné příznaky jsou vyjmenovány v tabulce 9.8. Nastavení PaX příznaku vyžaduje superuživatelské oprávnění. Syntaxe příkazu je

```
paxctl <volby-(čili-nastavované-příznaky)> <ovlivňované-soubory>
```

11.4.2.3 Ověření funkčnosti PaX v systému

`paxtest` je utilita určená k otestování funkčnosti konfigurace PaX [AddUti]. Provádí činnosti, kterým by měl správně nakonfigurovaný PaX zamezit. Mezi ně patří pokusy o spuštění kódu ve všech možných mapováních, pokusy o simulaci return-to-lib útoku a další podobné operace. Po každém nasazení PaX je jím vhodné otestovat systém. Jak může takové testování dopadnout je vidět na výstupech 11.2 a 11.3.

Výsledek je způsoben především tím, že `gcc` standardně netvoří spustitelné soubory typu etdyn, nýbrž etexec. Je-li zapnuta randomizace AS u etexec – RANDEXEC – potom jsou PaXem chráněny pouze ty spustitelné soubory, u kterých to ochranu explicitně vyžádáme příkazem `paxctl`.

```
$ paxtest kiddie
Executable anonymous mapping             : Killed
Executable bss                           : Killed
Executable data                          : Killed
Executable heap                          : Killed
Executable stack                         : Killed
Executable shared library bss            : Killed
Executable shared library data           : Killed
Executable anonymous mapping (mprotect)  : Vulnerable
Executable bss (mprotect)                : Vulnerable
Executable data (mprotect)               : Vulnerable
Executable heap (mprotect)               : Vulnerable
Executable stack (mprotect)              : Vulnerable
Executable shared library bss (mprotect) : Vulnerable
Executable shared library data (mprotect): Vulnerable
Writable text segments                   : Vulnerable
Anonymous mapping randomisation test     : 28 bits (guessed)
Heap randomisation test (ET_EXEC)        : 14 bits (guessed)
Heap randomisation test (PIE)            : 28 bits (guessed)
Main executable randomisation (ET_EXEC)  : 29 bits (guessed)
Main executable randomisation (PIE)      : 28 bits (guessed)
Shared library randomisation test        : 29 bits (guessed)
Stack randomisation test (SEGMEXEC)      : 30 bits (guessed)
Stack randomisation test (PAGEEXEC)      : 30 bits (guessed)
Arg/env randomisation test (SEGMEXEC)    : 22 bits (guessed)
Arg/env randomisation test (PAGEEXEC)    : 22 bits (guessed)
Randomization under memory exhaustion @~0: 28 bits (guessed)
Randomization under memory exhaustion @0 : 29 bits (guessed)
Return to function (strcpy)              : paxtest: return address contains a NULL byte.
Return to function (memcpy)              : Return to function (strcpy, PIE)        : paxtest: return
↪  address contains a NULL byte.
Return to function (memcpy, PIE)         :
```

Příkaz/y 11.2: Výstup nástroje `paxtest` na systému prostého PaX

11.4.2.4 Ověření nastavení PaX pro jednotlivé subjekty

Pro ověření nastavení PaX slouží příkaz `pspax`. Je schopný vypsat v systému běžící procesy společně s příznaky Grsec/PaX je ovlivňujících. Příklad výstupu nástroje `pspax` je uveden v příkazu 11.4.

```
$ paxtest
Executable anonymous mapping             : Killed
Executable bss                           : Killed
Executable data                          : Killed
Executable heap                          : Killed
Executable stack                         : Killed
Executable anonymous mapping (mprotect)  : Killed
Executable bss (mprotect)                : Killed
Executable data (mprotect)               : Killed
Executable heap (mprotect)               : Killed
Executable shared library bss (mprotect) : Killed
Executable shared library data (mprotect): Killed
Executable stack (mprotect)              : Killed
Anonymous mapping randomisation test     : 16 bits (guessed)
Heap randomisation test (ET_EXEC)        : 13 bits (guessed)
Heap randomisation test (ET_DYN)         : 25 bits (guessed)
Main executable randomisation (ET_EXEC)  : 16 bits (guessed)
Main executable randomisation (ET_DYN)   : 17 bits (guessed)
Shared library randomisation test        : 16 bits (guessed)
Stack randomisation test (SEGMEXEC)      : 23 bits (guessed)
Stack randomisation test (PAGEEXEC)      : 24 bits (guessed)
Return to function (strcpy)              : Vulnerable
Return to function (strcpy, RANDEXEC)    : Killed
Return to function (memcpy)              : Vulnerable
Return to function (memcpy, RANDEXEC)    : Killed
Executable shared library bss            : Killed
Executable shared library data           : Killed
Writable text segments                   : Killed
```

Příkaz/y 11.3: Výstup nástroje `paxtest` na systému s nasazeným PaX. Zde si lze povšimnout detekovaných zranitelností vůči útoku „return to libc“

```
# pspax
USER     PID    PAX    MAPS ETYPE      NAME            CAPS ATTR
root     1      PeMRs  w^x  ET_DYN     init            =ep
root     935    PeMRs  w^x  ET_DYN     udevd           =ep
root     1371   PeMRs  w^x  ET_DYN     udevd           =ep
root     1372   PeMRs  w^x  ET_DYN     udevd           =ep
root     1555   PeMRs  w^x  ET_DYN     syslogd         =
root     1597   PeMRs  w^x  ET_DYN     klogd           =
root     1683   PeMRs  w^x  ET_DYN     crond           =ep
root     1689   PeMRs  w^x  ET_DYN     ntpd            =cap_net_bind_service,cap_sys_time+ep
root     1703   PeMRs  w^x  ET_DYN     login           =ep
root     1704   PeMRs  w^x  ET_DYN     mingetty        =ep
root     1705   PeMRs  w^x  ET_DYN     mingetty        =ep
root     1706   PeMRs  w^x  ET_DYN     mingetty        =ep
```

Příkaz/y 11.4: Výstup nástroje `pspax`, zobrazující volby, s nimiž běží jednotlivé procesy v rámci systému

12. Dokumentace

V této kapitole se čtenář seznámí se sadou funkcí, která byla použita pro získání a zpracování výsledků v tomto dokumentu prezentovaných. Všechny tyto funkce jsou umístěny v jediném souboru umístěném na dále označovaným jen slovem „Skript“. Skript lze charakterizovat těmito body:

- Je napsán v jazyce BASH 4.3.
- Je z zvětší části neinteraktivní.

  Skript během svého vykonávání neinteraguje s uživatelem s výjimkou situace, kdy se nedaří zdárně dokončit nějaký z testů a existuje šance, že by to mohl změnit zásah uživatele – např. spuštěním serverové části testu.

12.1 Obsah přiloženého CD

Obsah přiloženého CD je zachycen obrázkem 12.1.

```
/
├── configurations ..... adresář s artefakty týkajícími se konfigurací měřených systémů
│   ├── configs ..... adresář obsahující konfigurační soubory vzniklé při konfiguraci jader
│   └── kernels ..... adresář obsahující jádra představující jednotlivé testované konfigurace
├── test-tools ..... adresář s testovacími nástroji nepřítomnými v balíčkovacím systému OS Debian 9.0
│   ├── lmbench3 ..... adresář s testovacím nástrojem lmbench verze 3
│   ├── qperf-0.4.9 ..... adresář s testovacím nástrojem qperf verze 0.4.9
│   └── unixbench ..... adresář s testovacím nástrojem unixbench verze 5.1.2
├── thesis-src ..... adresář obsahující zdrojové soubory, z nichž čerpá tento dokument
│   ├── bargraphs ..... adresář s grafy uvedenými v kapitole 14.1
│   ├── results ..... adresář s výsledky měření
│   └── tex ..... adresář s textovými soubory zakomponovanými v těle tohoto dokumentu
│       ├── part.teorie.tex ..... stěžejní část tohoto dokumentu ve formátu XeLaTeX
│       └── ⋮ ..... řada dalších zdrojových souborů ve formátu XeLaTeX
├── thesis.pdf ..... elektronická verze tohoto dokumentu
└── script ..... adresář obsahující funkce použité pro získávání výsledků
    ├── prog ..... soubor obsahující funkce provádějící testování a funkce generující výstupy z tohoto testování
    └── conf ..... konfigurace chování funkcí obsažených v souboru prog
```

Obrázek 12.1: Stav FS umístěného na přiloženém CD z pohledu jeho kořenového adresáře

12.2 Programátorská dokumentace

Všechen kód je až na výjimky vykonáván shellem nastaveném dle výpisu uvedeném v příkazu 12.1. Během získávání hodnot z testů nelze předpokládat o spouštěných testech absolutně nic. Některé testy posílají naměřené výsledky výhradně do `/dev/stderr`. Některé testy vracejí nulové návratové kódy i v situaci, kdy jejich vykonávání selhalo. V takovém případě je zapotřebí jejich selhání detekovat jinak, nežli jen analýzou jejich návratové hodnoty. Obecně jakékoliv zpracování výstupu skriptem volaného programu je nakonec předmětem individuální konfigurace pro ten který program včetně reakcí na různé návratové kódy.

Proto Skript vykonává testy v prostředí uvedeném v ukázce kódu 12.1, ve kterém jsou odnastaveny atributy pipefail a errexit. Vysvětlení významu proměnných vyskytnuvších se ve skriptu 12.1 je uvedeno v tabulce 12.1.

12.2.1 Dva přístupy k ukončení funkce

Při návrhu skriptu byl kladen důraz především na jeho robustnost. Funkce v jazyce BASH 4.3 nedisponují mechanismem, který by po jejich ukončení automaticky smazal dočasné soubory a při jakémkoliv chybovém stavu je toto třeba provést ručně. Není-li daná činnost automatizována, je snadné na ni zapomenout. Proto ve Skriptu existuje nikoliv nepodstatná množina funkcí, jejichž volání odpovídá skriptu 12.2. Do jejich spuštění je vklíněna funkce `fn_subshell`, která má **zajistit nastavení prostředí pro volanou funkci** (dále již stále „volaná funkce“) a zároveň se snaží být maxi-

```
$ set +o
set +o allexport
set -o braceexpand
set +o emacs
set -o errexit
    \ Skončí jakmile nějaký příkaz skončí nenulovým návratovým kódem.
set -o errtrace
    \ Nastavení trap pro ERR děděno do funkcí shellu.
set +o functrace
set -o hashall
set +o histexpand
set +o history
set +o ignoreeof
set -o interactive-comments
set +o keyword
set +o monitor
set +o noclobber
set +o noexec
set +o noglob
set +o nolog
set +o notify
set -o nounset
    \ Při substituci považuje nenastavené proměnné za chybu.
set +o onecmd
set +o physical
set -o pipefail
    \ Návratová hodnota kolony je status posledního příkazu, který skončil s nenulovým kódem.
set +o posix
set +o privileged
set +o verbose
set +o vi
set +o xtrace
```

Příkaz/y 12.1: Nastavení shellu při vykonávání daného skriptu. Oproti „výchozímu nastavení“ změněné volby jsou okomentovány.

```
trap - ERR          # deaktivace volání (konkrétně fn. "fn_trap_err") v případě výskytu !0 návratové
 ↪  hodnoty
set +o pipefail     # => návratová hodnota kolony == návratová hodnota posledního jejího příkazu
set +o errexit      # => !0 návratová hodnota nezpůsobí konec příkazového interpretru
eval "( PATH=\"${PATH_UPDATE}\" ; export ${VAR_FOR_EXPORT} ; auxfn_catret ${RET} <\"${IN}\" | ${CMD}
 ↪  1>\"${OUT}\" 2>\"${ERR}\" ; \
                          echo \"\${PIPESTATUS[@]}\" >${PIPERET} )"
set -o pipefail     # => návratová hodnota kolony == návratová hodnota prvního příkazu v ní, který
 ↪  vrátí !0 hodnotu
set -o errexit      # => příkazový interpret skončí jakmile jakýkoliv proces jím spuštěný vrátí !0
 ↪  hodnotu
${TRAP_BACKUP}      # aktivace volání (konkrétně fn. "fn_trap_err") v případě výskytu !0 návratové
 ↪  hodnoty
```

Skript 12.1: Prostředí pro vykonávání testujícího programu, jehož jméno je uloženo v `${CMD}`

Proměnná	Význam
PATH_UPDATE	Nová hodnota proměnné prostředí PATH. Její hodnotu je vhodné měnit v případě, je-li testem spustitelný soubor či skript získaný stažením (a kompilací) čili obecně získaný mimo balíčkovací systém a není tak přítomen v umístěních definovaných v ${PATH} či tam přítomen je, avšak v jiné, nežli požadované verzi.
VAR_FOR_EXPORT	Proměnná obsahující řetězec <VAR>=<VAL> [<VAR>=<VAL> ..] pro případný export proměnných, které jsou nutné pro běh příkazů. Vstupy formou proměnných prostředí nebyly při návrhu v podstatě vůbec uvažovány a proto také nebyly dostatečně rozmyšleny. Z toho důvodu je toto řešení poměrně nesystémové.
RET	Návratová hodnota předchozího příkazu v „pipeline“. Funkce auxfn_catret() zde představuje „replikaci“ standardního výstupu a návratové hodnoty předchozího příkazu.
IN	Standardní výstup předchozího příkazu „pipeline“.
CMD	Vykonávaný příkaz či posloupnost příkazů.
OUT	Název dočasného souboru pro zachytávání standardního výstupu.
ERR	Název dočasného souboru pro zachytávání standardního chybového výstupu.
PIPESTATUS	Proměnná nesoucí návratové hodnoty každého z příkazů poslední roury (zřetězení příkazů) běžící na popředí.
PIPERET	Název dočasného souboru pro zachytávání obsahu proměnné PIPESTATUS.

Tabulka 12.1: Význam proměnných vyskytujících se v „prostředí pro vykonávání kódu“

málně transparentní. Toto nastavení pochopitelně není samozřejmé[1]. Sestává z následujících skutečností:

```
function <NÁZEV-FUNKCE> { fn_subshell '' UNPROTECTED_<NÁZEV-FUNKCE> "${@:-}" ; }
function UNPROTECTED_<NÁZEV-FUNKCE>
{
    # ...
}
```

Skript 12.2: Formát definice řady funkcí umístěných ve Skriptu

1. **Volaná funkce běží v shellu, který je pro její běh speciálně vytvořen (a přednastaven)**.

   Tento subshell je volán v těle funkce fn_subshell. Poté je inicializován (viz body 2, 3, 4 a 5) a nakonec[3] je v něm spuštěna „volaná funkce“.

2. **Chování shellu je nastaveno dle příkazu[5] 12.1**.

   Nastavením hodnot atributů shellu je zapotřebí dosáhnout požadovaného nastavení[6] běhového prostředí pro volanou funkci.

3. **Je vytvořen adresář pro uložení dočasných souborů vzniklých za běhu volané funkce**.

   Funkce fn_subshell volá
   ```
   mktemp -d
   ```
   . Výstupem tohoto volání je název adresáře, který mktemp vytvořil v dočasném adresáři. Výstup se obratem ukládá do proměnné ${TMPDIR}, která je exportována do subshellů. Jelikož mktemp vytváří dočasné soubory v adresáři stanoveném proměnnou ${TMPDIR}, je-li nastavena, je tímto zaručeno, že strom adresářů s dočasnými soubory bude kopírovat strom volání funkce fn_subshell.

4. **Nastavení volání při zachycení signálu EXIT**.

   Funkce fn_subshell již v rámci subshellu, v němž bude vykonávána volaná funkce, volá
   ```
   trap "${SUBSHELL_FN_TRAP}" EXIT
   ```

[2] Nelze se spolehnout na to, že shell, v němž budou funkce skriptu vykonávány bude nastaven stejně a proto je zapotřebí jej explicitně nastavit.

[4] Proto návratová hodnota „volané funkce“ představuje návratovou hodnotu celého subshellu. Přenos návratové hodnoty je tak navenek „transparentní“.

[5] To dlouho neplatilo až tak docela. Funkce fn_subshell měnila volby různé oproti výchozímu nastavení. Lepším řešením však je nastavení všech voleb, neboť není možné předjímat, jaké má uživatel, který ji spustil, výchozí nastavení shellu. Ve finální verzi skriptu bylo toto napraveno.

[6] Nastavení se do subshellu z rodičovského shellu totiž nedědí.

[7] Dočasným adresářem bývá zpravidla adresář /tmp.

čímž říká, že při ukončování aktuálního shellu nechť je volán příkaz/funkce, jehož jméno je uloženo v `${SUBSHELL_FN_TRAP}`. V této proměnné je uložen název funkce provádějící úklid. Při ukončení shellu, v němž volaná funkce běžela (či běží) přichází standardně signál EXIT. Funkce `fn_subshell` změní výchozí stav, kdy se nevolá žádný příkaz, na stav, kdy je volána funkce `fn_trap_exit`, která smaže adresář, jehož jméno je uloženo v proměnné `${TMPDIR}`. Vzhledem k tomu, že strom adresářů s dočasnými soubory kopíruje strom volání funkcí, je při selhání kterékoliv funkce v rámci stromu smazán všechen obsah jí vytvořený včetně obsahu vytvořeného jejími potomky.

5. **Nastavení volání při zachycení signálu ERR**.

   Signál ERR přichází při prvním nenulovém návratovém kódu vyskytnuvšího se ve vykonávání[8]. Funkce `fn_subshell` nastavuje jako reakci na tento signál volání funkce `fn_trap_err`. Tato funkce je tak volána pokaždé, když vykonávání v shellu skončí nenulovým návratovým kódem některého z příkazů.

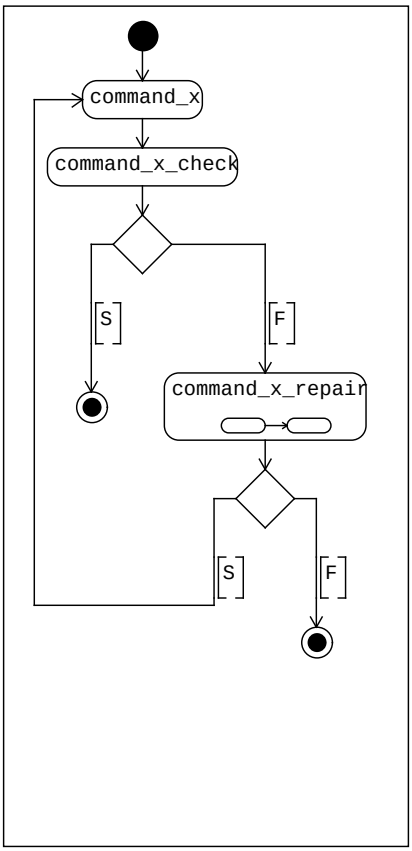
Příkaz volaný prostřednictvím funkce `fn_subshell` tak oproti příkazu volanému přímo

- neponechá v externí paměti žádné zapomenuté dočasné soubory,
- skončí při prvním výskytu nenulového návratového kódu,
- bude mít jasně definované podmínky dané mu nastavením atributů shellu
- a při nezdárném konci budou vypsány na `/dev/stderr` poslední tři volání vedoucí v zásobníku volání k selhavšímu příkazu.

12.2.2 Implementované funkce

Funkce implementované ve Skriptu jsou opatřeny komentáři. Čtenář si je může prohlédnout v souboru `/script/proc` umístěném na přiloženém CD. Většinou se jedná o funkce natolik rozsáhlé, že by nebylo vhodné je zde prezentovat nebo naopak tak drobné, že jejich ukázka by se bez dlouhosáhlého popisu kontextu tvořeného ostatními funkcemi neobešla.

12.3 Uživatelská



Obrázek 12.2: Základní blok pipeline získávající hodnotu

[9] Za předpokladu, že je shellu předtím nastaven atribut `errexit`.

Skript provádějící testování – `/prog` – sestává z množiny funkcí, které je pochopitelně možné přímo spouštět. U většiny z nich to však nemá význam, neboť jejich spuštění zpravidla předpokládá existenci „kontextu“ – např. přítomnost souborů na určitých adresách FS či předchozí nastavení proměnných. Předpokládá se, že jakákoliv činnost, k níž skript poskytuje prostředky, bude uživatelem vyvolána voláním funkce `_main()` s příslušnými parametry. Výčet těchto parametrů je uveden v tabulce 12.2. Funkce programem používané lze buďto importovat do shellu či spustit v samostatném shellu – viz příkazy 12.2.

```
$ . prog            # import definic funkcí do aktuálního shellu
$ _main -h          # spuštění funkce '_main' s parametrem vynucujícím vypsání nápovědy

$ ./prog _main -h   # totéž, avšak v samostatném shellu (subshellu)
```

Příkaz/y 12.2: Příkazy iniciující chod programu. V tomto případě pouze vypsání nápovědy k ovládání programu.

12.3.1 Konfigurační soubor

```
[section1]
option1 = value1
option2 = value2
# comment

[section2]
option1 = value3
```

Konfigurace 12.1: Syntaxe skriptem používaného konfiguračního souboru

Skript `_prog` ke svému běhu vyžaduje přítomnost konfiguračního souboru, neboť výhradně prostřednictvím něj[10] lze Skriptu zadat většinu parametrů, dle kterých potom koná. Používá formát INI, jehož syntaxe je naznačena v konfiguraci 12.1. Každé přiřazení umístěné v konfiguračním souboru je transformováno v přiřazení hodnoty do lokální proměnné v jazyce BASH. Sekce obsažené v konfiguračním souboru lze rozlišit na

- **sekce nesoucí obecné konfigurace** a

  Pojmenování těchto sekcí nezačíná číslem. Sekce `[configurations]` obsahuje množinu všech možných pojmenování konfigurací systému. Ta je používána pro prevenci zadání chybných pojmenování. Sekce `[script-settings]` obsahuje nastavení, která jsou globální napříč celým Skriptem čili pro všechna měření.
- **sekce nesoucí konfigurace určené ke zřetězení**.

  Pojmenování těchto sekcí začíná číslem. Příklad sekce nesoucí konfiguraci ke zřetězení je vyobrazen v konfiguraci 12.2. Číslo v jejím názvu je podstatné. Udává totiž pořadí, v jakém se budou jednotlivé sekce nesoucí konfigurace určené ke zřetězení respektive konfigurace jimi nesené aplikovat.

```
[1100,iperf_local]
run_on                  = tested
counterparty            = local
SERVERNODE              = localhost
```

Konfigurace 12.2: Příklad sekce nesoucí konfiguraci ke zřetězení

Na přiloženém CD se nachází konfigurační soubor v podobě, v jaké byl použit při získávání dat uvedených v tomto dokumentu.

12.3.1.1 Sdílení hodnot mezi měřeními

Při návrhu skriptu vyvstal **problém sdílení hodnot mezi jednotlivými měřeními**, neb měření se mnohdy liší jen v drobných detailech jako je např. použitý přenosový protokol. Cílem bylo vymyslet způsob, jak co nejjednodušeji sdílet různá nastavení různých měření. Po neúspěšných pokusech s asociativními poli bylo nalezeno prosté řešení. Toto řešení je postavené na skutečnosti, že každé nastavení hodnoty proměnné přepíše dřívější nastavení její hodnoty. Proto jsou konfigurace uspořádány do hierarchické struktury číselných hodnot zapisovaných v desítkové soustavě a aplikuje se nad nimi následující postup:

[11] Toto řešení se záhy ukázalo jako dostatečně nepružné. Požadavek na změnu souboru při každém nepatrném ladění parametrů byl navýsost život otravující. Lépe by zřejmě bylo interně implementovat transformaci záznamů v INI souboru do podoby parametrů příkazu, kde parametry vygenerované z obsahu INI souboru by měly nižší prioritu nežli parametry předané klasickou cestou. Například by byly jen uvedeny dříve a proto jimi prováděné změny by mohly být pozdějšími řádnými parametry předefinovány. Leč toto nebylo promyšleno dostatečně včas a na pozdější úpravy již nebyl čas.

Nechť x je číslo měření, které má být vykonáno.

1. Aplikují se všechna přiřazení ze sekce s pořadovým číslem 0.
2. Aplikují se všechna přiřazení ze sekce s nejmenším pořadovým číslem s toutéž hodnotou nejvyššího řádu, jako má číslo x a zároveň s pořadovým číslem menším x.
3. Právě aplikovaná sekce se v příštím kroku ignoruje. Neexistuje-li žádná další konfigurace, která by se mohla v bodě 2 aplikovat, potom se pokračuje bodem 4. Jinak se pokračuje se bodem 2.
4. Aplikují se všechna přiřazení ze sekce s pořadovým číslem x[12].

Mimo nečíslovaných sekci konfigurační soubor obsahuje dva typy sekcí číslovaných:

koncové sekce a
: Jejich číselný kód vždy končí nenulovou cifrou. Představují poslední konfiguraci ve zřetězení konfigurací. V nich obsažené nastavení voleb má nejvyšší možnou prioritu.

konfigurační sekce.
: Jejich číselný kód končí nulovou cifrou. Představují konfigurace na začátku/uprostřed řetězce konfigurací. Nastavení voleb v nich obsažené není definitivní, čili může být přepsáno v konfiguracích s vyšší prioritou.

Dejme tomu, že v konfiguračním souboru jsou přítomny konfigurace 0, 1, 1000, 1010, 1100, 1101, 1102. Při požadavku na provedení testu 1102 se aplikují konfigurace v tomto pořadí:

1. 0 – výchozí konfigurace
   se aplikuje vždy pro každé měření,
2. 1000 – obecná konfigurace pro testy iperf
   se aplikuje jako druhá, neboť má tentýž nejvyšší řád a zároveň je nejmenší,
3. 1100 – obecná konfigurace pro iperf na lokálním stroji
   se aplikuje jako třetí, neboť má tentýž druhý nejvyšší řád a zároveň je nejmenší,
4. 1102 – konfigurace konkrétního testu
   se aplikuje jako poslední a test se provede.

Jednotlivé volby obsažené v konfiguračním souboru představují potenciální definice proměnných v jazyce BASH. Skript si dle zadání postupně vybírá jednotlivé sekce a nastavuje všechny v nich obsažené proměnné. Tento princip umožnil sdílení hodnot mezi různými sekcemi. V případě skriptu `prog` je předpokládána existence sekcí configurations a script-settings. Sekce configurations je skriptem používána pro překlad

$$\text{pojmenování konfigurace} \rightarrow \text{kód konfigurace}$$

Před prováděním každého testu se ověřuje, pro jakou konfiguraci bude daný test prováděn. Pokud pro pojmenování konfigurace (např. „original“) neexistuje kód, konfigurace není programu známa. Tímto dojde k odhalení překlepů, které by vedly k pracným opravám v budoucím zpracování naměřených výsledků.

12.3.2 Postup práce se skriptem

Funkce skriptu `_main` poskytuje uživateli podporu od sběru hodnot přes jejich úpravu až po konečnou prezentaci. Činnost, kterou funkce `_main` započne, určuje uživatel prostřednictvím přepínačů vyjmenovaných v tabulce 12.2.

Následuje podrobnější popis jednotlivých akcí.

12.3.2.1 Akce iniciovaná přepínačem `--test`

Nejvýznamnější činností, kterou Skript vykonává je získávání výsledků, které je iniciováno přepínačem `--test`. Pro každou koncovou sekci, jejíž kód byl zadán prostřednictvím přepínače `--list` provede skript právě jedno volání funkce `fn_gen_test()`. V rámci ní se postupně volají příkazy uložené ve vybraných proměnných definovaných v konfiguraci daného testu. Tyto proměnné, vyjmenované v tabulce 12.3 a reprezentující jednotlivé úrovně zpracování měřením získaných hodnot v pipeline, mohou nést

- **volání funkce definované v aktuálním prostředí**,
  Funkce definované v rámci shellu, v němž probíhá testování představují vždy výchozí hodnotu v případě nenastavení proměnné. Těmito funkcemi jsou
  - default snažící se „nedělat nic“ pouhým zkopírováním výstupu a návratové hodnoty jí předcházejícího programu/funkce v pipeline,

[12] Pokud taková neexistuje, potom měření nebude provedeno.

Přepínač	Očekávaná hodnota	Dopad na běh skriptu
`-h, --help`		Skript vypíše nápovědu a skončí.
`--cfg`	cesta ke konfiguračnímu souboru	Skript konfigurační soubor potřebuje ke každé své činnosti vyjma vypsání nápovědy a vypsání seznamu dostupných testů.
`--offer`		Skript vypíše seznam dostupných konfigurací (testů).
`--list`	seznam testů	Program vykoná všechny akce pro všechny testy uvedené v tomto seznamu. Seznam může obsahovat prvky oddělené čárkami. Těmito prvky mohou být • čísla (např. „105“) či • dvojice čísel oddělených pomlčkou (např. „108-112“). Seznam testů ve tvaru „105,108-112“ znamená požadavek na vykonání všech požadovaných akcí pro testy 105, 108, 109, 110, 111 a 112.
`--confname`	pojmenování konfigurace	Skript vyžaduje znát jméno konfigurace, pod kterou provádí testování. Jméno konfigurace je tak nutné pouze v případě uvedení přepínače `--test`.
`--test`		Skript provede testování všech testů ze seznamu testů.
`--csv`		Skript z výsledků jednotlivých testů sestaví csv soubory pro všechny testy ze seznamu testů.
`--graph`		Skript vygeneruje grafy z csv souborů pro všechny testy ze seznamu testů.
`--tex`		Skript vygeneruje výstup pro sazbu grafů a tabulek s výsledky v LaTeXu.
`--proc`		Program se chová tak, jako kdyby byly zadány přepínače `--csv`, `--graph` a `--tex` zároveň.

Tabulka 12.2: Argumenty akceptované programem

  - default_check provádějí testování, zda-li výstup z předchozího programu/funkce je správný či nikoliv dle zadaných přepínačů a
  - default_repair pokoušející se o opravu chybového stavu.

- **volání spustitelného souboru dostupného v rámci FS** nebo

  Do proměnných vyjmenovaných v tabulce 12.3 lze pochopitelně ukládat i volání běžných souborů ležících na FS.

- **něco úplně jiného**.

  Pokud jakákoliv z proměnných vyjmenovaných v tabulce 12.3 nebude obsahovat validní volání, test skončí nezdarem.

Tabulka 12.3: Proměnné nesoucí volání vykonávané v pořadí, jakém jsou uvedeny v tabulce

Jméno proměnné	Význam volání uloženého v dané proměnné
`command_prepare`	Příprava prostředí pro test. Přípravou může být například zjištění, zda-li je protistrana dostupná (jedná-li se o test sítě) či zda-li existuje na FS soubor, který během testu existovat má čili s jehož existencí daný test počítá. Výchozí hodnota: `default` Volání prováděno: vždy
`command_prepare_check`	Zjištění, zda-li volání `command_prepare` proběhlo úspěšně. V případě nezdaru se dále volá funkce `command_prepare_repair`, aby se nezdar pokusila odstranit.[14] Výchozí hodnota: `default_check --err-ecode '[1-9]+'` Volání prováděno: vždy

↓ Pokračování tabulky na další straně.

Pokračování tabulky z předchozí strany. ↑

Jméno proměnné	Význam volání uloženého v dané proměnné
`command_prepare_repair`	V případě že návratový kód funkce/program obsažený v proměnné `command_prepare_check` bude nenulový, bude volána funkce/program, jejíž název je uložen v proměnné `command_prepare_repair`. Výchozí hodnota: `default_repair` Volání prováděno: pouze v situaci, kdy volání uložené v `command_prepare_check` skončilo nezdarem
`command_test`	Název funkce/programu provádějící samotný sběr dat čili měření. Výchozí hodnota: `default`, nicméně nemá význam Volání prováděno: vždy
`command_test_check`	Funkce/program, jehož název je uložen v proměnné `command_test_check` je určena ke kontrole, zda-li testovací funkce/program z `command_test` proběhla správně či nikoliv. Výchozí hodnota: `default_check --err-ecode '[1-9]+'`[14] Volání prováděno: vždy
`command_test_repair`	Pokus o opravu „prostředí“ v případě, že test skončil nezdarem. Výchozí hodnota: `default_repair` Volání prováděno: pouze v situaci, kdy volání uložené v `command_test_check` skončilo nezdarem
`command_clean`	Úklid prostředí po testu. Volání by mělo zajistit smazání souborů vzniklých během testování na FS. Výchozí hodnota: `default` (v tomto „stavu“ se žádný úklid nekoná) Volání prováděno: vždy
`command_clean_check`	Kontrola, zda-li úklid prostředí proběhl správně. Výchozí hodnota: `default_check --err-ecode '[1-9]+'`[14] Volání prováděno: vždy
`command_clean_repair`	Pokus o opravu v případě, že úklid prostředí skončil nezdarem. Výchozí hodnota: `default_repair` Volání prováděno: pouze v situaci, kdy volání uložené v `command_test_check` skončilo nezdarem
`command_postproc_crop`	Volání zajišťující zpracování výstupu testu do jedné hodnoty. Cílem by mělo být vždy získání (typicky dekadického) čísla s ukončeným rozvojem. Výchozí hodnota: `default_repair` Volání prováděno: pouze v situaci, kdy volání uložené v `command_test_check` skončilo nezdarem

Konec tabulky.

Vstupy a výstupy těchto volání jsou vzájemně zřetězeny[13] a tvoří tak v podstatě simulovanou rouru. Po provedeném testu na FS vznikne v adresáři adresářová struktura vyobrazená na obrázku 12.3.

12.3.2.2 Akce iniciovaná přepínačem `--csv`

Přepínač `--csv` předaný funkci `_main` způsobí, že se již naměřené hodnoty

1. **převedou do požadované jednotky**,

   Požadovanou jednotku specifikuje uživatel v konfiguračním souboru (viz kapitola 12.3.1). Na FS již uložené hodnoty však zůstanou beze změny v původní jednotce. Převod do jiné jednotky proběhne pouze v paměti

[14] Pořadí zřetězení techto volání je stejné jako pořadí proměnných v tabulce 12.3.

[14] Při návrhu *skriptu* se počítalo s tím, že opravy budou různého druhu a budou spočívat v různých krocích. Nakonec jsou výsledkem tři druhy oprav: „Oprava odstraněním výsledku.“, „Oprava zásahem (lidské) obsluhy.“ a „Žádná oprava.“

[14] Návrh *skriptu* počítal s tím, že cokoliv se „může nezdařit“ a proto skript musí být dostatečně robustní. U samotného testovacího příkazu umoženého v proměnné `command_test` jeho případný nezdar dává smysl a proto jsou `command_test_check` a `command_test_repair` na místě. Aby byl skript dostatečně obecný, princip „kontroly a opravy“ se rozkopíroval do všech možných úrovní včetně „přípravy prostředí“ a „úklidu prostředí“. Konfigurace těchto možností však ani zdaleka nevyužívá a obecnost při psaní skriptu zde tak je spíše na škodu a obtížnější orientaci v něm, nežli k užitku samému.

[14] Díky nahraditelnosti volání lze úspěch definovat zcela jinak než jen jako „nulový návratový kód“.

[14] Proměnná, jejíž hodnota je modifikovatelná prostřednictvím konfiguračního souboru.

a původní naměřené hodnoty tak nepostihne[15].

2. **umístí do tabulky ve formě CSV souboru**,

   CSV soubor takto vzniklý je umístěn v kořenovém adresáři pro výstup skriptu. Jeho pojmenování je `/<testcode>.abs.csv`.

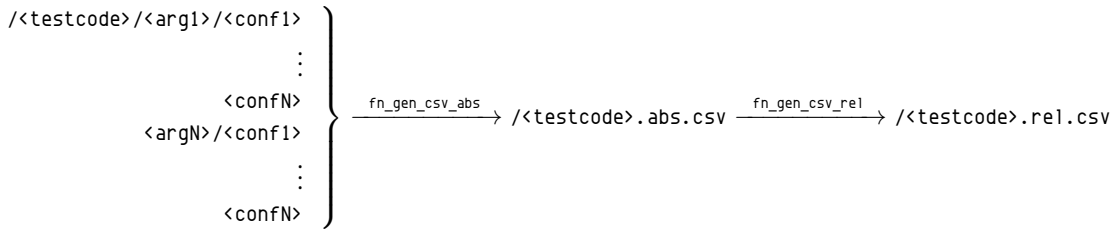
3. **přepočtou na relativní vzhledem k referenční konfiguraci** a

   Za referenční se považuje k. original. Hodnoty naměřené pro ni budou v relativním svém vyjádření tak představovat vždy hodnotu 1. Hodnoty naměřené pro ostatní konfigurace budou k této hodnotě přepočteny.

4. **tyto relativní umístí do druhého CSV souboru**.

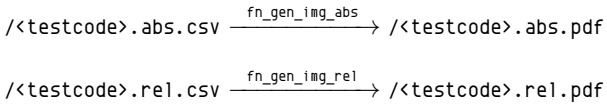
   CSV soubor takto vzniklý je umístěn v kořenovém adresáři pro výstup skriptu. Jeho pojmenování je `/<testcode>.rel.csv`.

Stručněji řečeno přepínač `--csv` způsobí následující:



12.3.2.3 Akce iniciovaná přepínačem `--graph`

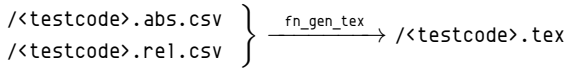
Přepínač `--graph` způsobí, že pro všechny testy vyjmenované přepínačem `--list` jsou volány funkce `fn_gen_img_abs` a `fn_gen_img_rel`, které provedou následující zobrazení:



Obě volají funkci `fn_gen_img`, která připravuje konfiguraci pro nástroj `gnuplot`, který následně volá. Jeho výstup je poté převeden z formátu SVG do formátu PDF nástrojem `convert`.

12.3.2.4 Akce iniciovaná přepínačem `--tex`

Přepínač `--tex` způsobí, že pro všechny testy vyjmenované přepínačem `--list` je volána funkce `fn_gen_tex`, která provede následující zobrazení:



Výstupem je soubor ve formátu TEX rozdělený na úseky oddělené vždy jedním řádkem obsahujícím zakomentovaný název následující „sekce“, např.

```
%------------------------------------------------------------------------ table-rel
\begin{table}[ht!]
    \centering
...
%------------------------------------------------------------------------ table-abs
...
```

Jednotlivé sekce tímto postupem vzniklé následně byly vkládány do tohoto dokumentu jako některé podkapitoly (viz kapitoly 13).

12.3.2.5 Akce proc

Přepínač `--proc` vyvolá tutéž reakci, jako kdyby byly funkci `_main` předány přepínače `--csv`, `--pdf` a `--tex` zároveň.

[15] Díky tomu je možné měnit prezentaci naměřených dat bez nutnosti opakovaného měření, neboť již jednou měřená data zůstávají stále stejná.

```
/ ..... kořenový adresář připojení do FS v bodě definovaném pozičním parametrem při spuštění skriptu
└── <testcode-1> ..... adresář obsahující všechny naměřené hodnoty
    ├── <argument-1> ..... adresář obsahující všechny naměřené hodnoty pro konkrétní parametr testu
    │   ├── <confname-1> ..... soubor obsahující naměřenou hodnotu v původní jednotce
    │   ├── ⋮
    │   └── <confname-n>
    ├── ⋮
    └── <argument-m>
        ├── <confname-1>
        ├── ⋮
        └── <confname-n>
```

Obrázek 12.3: Stav FS z pohledu adresáře, do něhož jsou ukládány hodnoty získané měřením po provedení testu „`<testcode-1>`“ pro všechny argumenty $\langle 1, m \rangle$ v konfiguracích $\langle 1, n \rangle$

```
/
├── <testcode-1>
│   ├── <argument-1>
│   │   ├── <confname-1>
│   │   ├── ⋮
│   │   └── <confname-n>
│   ├── ⋮
│   └── <argument-m>
│       ├── <confname-1>
│       ├── ⋮
│       └── <confname-n>
├── <testcode-1>.abs.csv ..... CSV obsahující všechny naměřené hodnoty převedené do požadované jednotky
├── <testcode-1>.rel.csv ..... CSV obsahující všechny naměřené hodnoty přepočtené k referenční hodnotě
├── <testcode-1>.abs.pdf ..... graf zobrazující data ze souboru /<testcode>.abs.csvve vizuální formě
├── <testcode-1>.rel.pdf ..... graf zobrazující data ze souboru /<testcode>.rel.csvve vizuální formě
└── <testcode-1>.tex ..... soubor ve fomrátu LaTeX vkládaný do tohoto dokumentu
```

Obrázek 12.4: Stav FS z pohledu adresáře, do něhož jsou ukládány hodnoty získané měřením, po zpracování hodnot z měření vzešlých

13. Testy

Kapitola 13 prezentuje výsledky měření. Pro jejich lepší pochopení se předpokládá znalost pojmů

- **závisle proměnná veličina**,

  Proměnná veličina, jejíž změny závisejí na dalších faktorech. Její hodnotu se snažíme nalézt. [Mec12]
- **nezávisle proměnná veličina** a

  Proměnná veličina, která se mění nezávisle vzhledem k jiným faktorům. Manipulujeme jí záměrně. [Mec12]

- **referenční systém/konfigurace**.

  Referenční systém či též *referenční konfigurace systému* představuje výchozí stav, vůči kterému jsou porovnávány naměřené hodnoty.

Měření jednotlivých veličin je možné provádět typicky opakovaným prováděním vlastního programu vykonávajícího určité elementární operace či specializovaného nástroje – benchmarku –, přičemž je zapotřebí

1. Zajistit neměnnost prostředí, ve kterém bude testování probíhat.
2. Stanovit veličiny, které jsou při provozu systému podstatné čili veličiny, které budou měřeny.
3. Pro každou veličinu:
   (a) Změřit její hodnotu na systému prostém jakýchkoliv úprav čili na systému s referenční konfigurací.
   (b) Změřit její hodnotu na systému s navrženými úpravami
   (c) Změřit její hodnotu na témže systému s
4. Naměřené výsledky vhodně vizualizovat.

13.1 Popis měření

Každá číselná hodnota prezentovaná v tomto dokumentu jako výsledek (v grafu či tabulce) byla získána vlastním měřením. Není-li v rámci kapitoly, kde je tato hodnota prezentována uvedeno jinak, potom platí následující tvrzení:

13.1.1 Získávání hodnot

- **Hodnota je aritmetickým průměrem z 5 naměřených hodnot.**

  Opatření vícenásobného měření má za cíl maximálně umenšit dopad možných odchylek při měření. 5 násobné opakování je v některých případech (13.3.1) málo a v některých případech (13.8.1) mnoho. Proto toto číslo nelze brát za konečné. Je pouze výchozím a může být pro konkrétní testy upraveno.
- **Hodnoty vzdálené od střední hodnoty o více jak 2 σ jsou z výpočtů vyřazeny.**

  Hodnoty měřené veličiny jsou získávány měřením, které je možné libovolně krát opakovat. Jedná se o získávání hodnot náhodné veličiny, které mohou být zatíženy chybami. Dle [17] je 95,45 % výsledků náhodně měřené veličiny ve vzdálenosti max. 2 σ od její střední hodnoty. Zbylé výsledky, vzdálené od střední hodnoty o více jak 2 σ, skript považuje za výsledky zatížené natolik velkou chybou, že je zahazuje a pokouší se získat další.

13.1.1.1 Postup získávání hodnot

Následující seznam zachycuje tok aktivit, tak jak je funkce `fn_gen_test` při měření veličiny X vykonává:

1. Test: Bylo získáno požadované množství hodnot?

   Ano. $\rightarrow$ Přejdi na bod č. 3.
2. Získání jedné další hodnoty – běh pipeline.

   Hodnota získána. $\rightarrow$ Přejdi na bod č. 1.
   Hodnota nezískána. $\rightarrow$ Přejdi na bod č. 7.
3. Z množiny hodnot měřené veličiny se vypočtou
   - střední hodnota $E(X)$ čili v tomto případě aritmetický průměr ze všech hodnot měřené veličiny a
   - směrodatná odchylka σ.
4. Pro každou jednu hodnotu se provede následující:

[0] Systém bez úprav bude v následné prezentaci výsledků figurovat jako systém referenční.

(a) Vypočte se maximální přípustná vzdálenost jedné hodnoty náhodné veličiny X od střední hodnoty této veličiny $E(X)$.
- vstup: $E(X)$, σ, hodnota zadaná v `${threshold_coeficient}`[14]
- výstup: hodnota leží/neleží v přípustné vzdálenosti

(b) Test: Leží hodnota v přípustné vzdálenosti?

Ne. → Hodnota se označí jako invalidní. V případě, že se v budoucnu nezmění střední hodnota, nebude s ní již dále počítáno.
Ano. → Hodnota se v množině naměřených hodnot ponechá.

5. Test: Je v množině naměřených hodnot požadované množství hodnot?

Ne. → Přejdi na bod č. 2.

6. Test byl vykonán úspěšně.
7. Test selhal. Na FS žádný výsledek uložen není.

13.1.2 Porovnávání hodnot

Hodnoty nezávisle proměnné veličiny se získávají pro různé *konfigurace testovaného prostředí* (viz kapitola 11.3). Takto získané uspořádané dvojice

$$(\text{hodnota nezávisle proměnné veličiny}, \text{konfigurace testovaného prostředí})$$

se následně vůči sobě porovnávají. Porovnání je relativní. Pro porovnání je nutné, aby byla získána hodnota pro referenční konfiguraci, to jest pro konfiguraci prostou všech úprav – konfiguraci „original“. Naměřené absolutní hodnoty jsou převedeny na koeficienty vyjadřující změnu oproti absolutní hodnotě naměřené v konfiguraci „original“.

Konfigurace original je stav měřeného prostředí, ve kterém je na STROJI XENMV jádro ve verzi.

Pro všechna uskutečněná měření platí,

- že navzájem porovnávaná měření probíhají za „týchž podmínek“ a
- že srovnání hodnot naměřených ukazatelů je vždy relativní vzhledem k referenci.

Je zřejmé, že absolutně tytéž podmínky není možné zaručit. Na testovacích strojích metalist a xenmv se během testování žádná činnost nesouvisející s měřením nevyvíjí. Určitou nestabilitu měřeného prostředí může přinést síťové spojení, které není v moci měření provádějící osoby.

13.2 Nástroje použité pro testování

Verze jednotlivých nástrojů použitých pro testování jsou uvedeny v tabulce 13.1.

Nástroj	Verze	Původ
iperf	2.0.5+dfsg1-2	Balíček iperf z repozitáře OS Debian 8.6.
ping	2:1.9.2.39.3a460-3	Balíček inetutils-ping z repozitáře OS Debian 8.6.
lmbench	3	Stažen z adresy `http://www.bitmover.com/lmbench/lmbench3.tar.gz`. Následně rozbalen a zkompilován. Výsledkem kompilace více samostatných binárních souborů provádějících testování.
unixbench	5.1.2	Stažen z adresy `http://github.com/kdlucas/byte-unixbench`. Následně rozbalen a zkompilován. Výsledkem kompilace více samostatných binárních souborů provádějících testování.
qperf	0.4.9	Stažen z adresy `https://www.openfabrics.org/downloads/qperf/qperf-0.4.9.tar.gz`. Následně rozbalen a zkompilován. Výstupem kompilace je jeden spustitelný soubor `qperf`, který zastane jak serverovou, tak i klientskou roli.
sysbench	0.4.12	Balíček sysbench z repozitáře OS Debian 8.6.

Tabulka 13.1: Verze nástrojů použitých pro testování

13.3 Testy nástrojem ping

13.3.1 Test ping

Test ping měří dobu od odeslání zprávy protistraně do přijetí odpovědi, čili obousměrné zpoždění. Nezávisle proměnnou je velikost datové části ICMP paketu. Testováno je od velikosti datové části 16 B, neboť právě to je minimální velikost, při které nástroj `ping` podá informaci o zpoždění. Je měřeno obousměrné zpoždění ICMP Echo Request/Reply paketů.

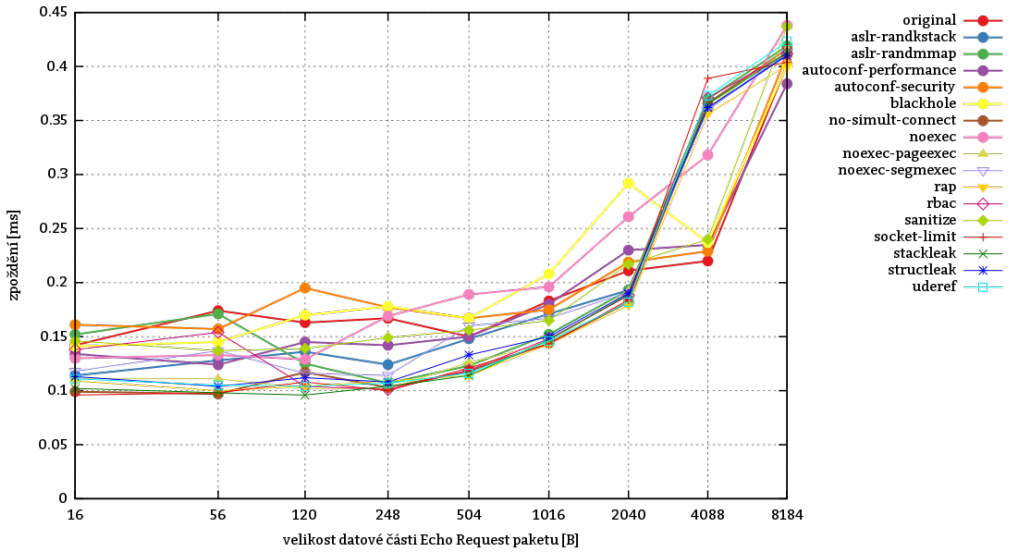
NZPV (x)	velikost datové části Echo Request paketu
jednotka NZPV	B
ZPV (y)	zpoždění
jednotka ZPV	ms
počet opakování měření	15
počítač spouštějící testovací příkaz	metalist
testovací příkaz	`ping -c 15 -s ${X} xenmv`

Tabulka 13.2: Charakteristiky testu ping

13.3.1.1 Naměřené hodnoty pro jednotlivé konfigurace

měřená konfigurace	velikost datové části Echo Request paketu [B]								
	16	56	120	248	504	1016	2040	4088	8184
original	0,142	0,174	0,163	0,167	0,15	0,183	0,211	0,22	0,402
aslr-randkstack	0,114	0,128	0,136	0,124	0,148	0,171	0,193	0,361	0,411
aslr-randmmap	0,152	0,171	0,125	0,107	0,123	0,152	0,193	0,37	0,419
autoconf-performance	0,134	0,124	0,145	0,142	0,15	0,18	0,23	0,235	0,384
autoconf-security	0,161	0,157	0,195	0,177	0,167	0,175	0,219	0,229	0,411
blackhole	0,141	0,145	0,17	0,178	0,167	0,208	0,292	0,237	0,401
no-simult-connect	0,099	0,097	0,117	0,103	0,118	0,144	0,182	0,366	0,416
noexec	0,13	0,133	0,129	0,169	0,189	0,196	0,261	0,318	0,438
noexec-pageexec	0,111	0,111	0,102	0,102	0,126	0,144	0,192	0,362	0,416
noexec-segmexec	0,118	0,137	0,116	0,114	0,16	0,167	0,191	0,36	0,413
rap	0,109	0,1	0,104	0,109	0,114	0,143	0,179	0,356	0,401
rbac	0,138	0,154	0,104	0,101	0,119	0,148	0,188	0,371	0,412
sanitize	0,146	0,137	0,139	0,149	0,156	0,165	0,217	0,24	0,437
socket-limit	0,096	0,098	0,108	0,1	0,121	0,143	0,184	0,389	0,404
stackleak	0,102	0,098	0,096	0,104	0,114	0,147	0,189	0,365	0,413
structleak	0,113	0,104	0,112	0,108	0,133	0,15	0,19	0,362	0,41
uderef	0,111	0,105	0,104	0,107	0,116	0,146	0,182	0,373	0,423

Tabulka 13.3: Absolutní srovnání konfigurací

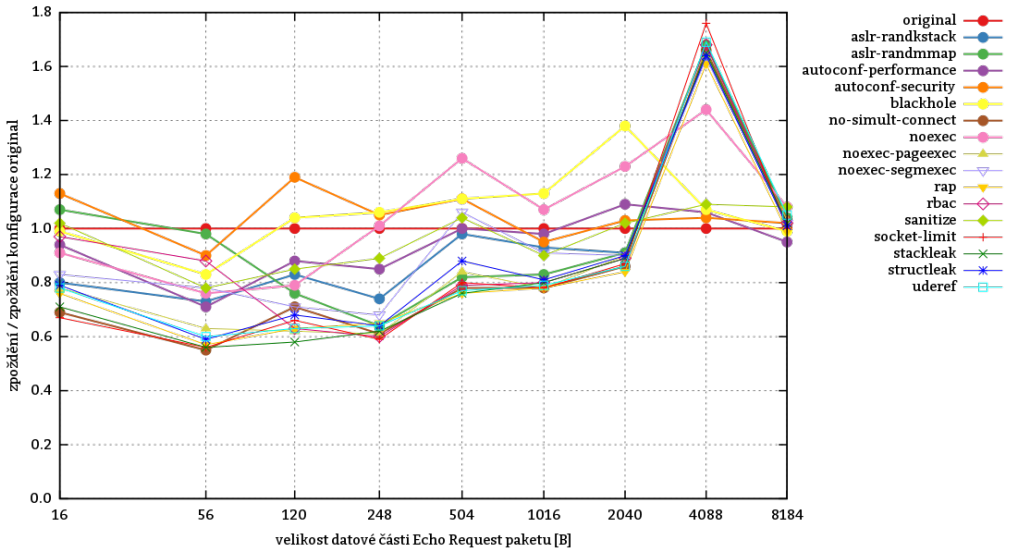


Obrázek 13.1: Absolutní srovnání konfigurací

13.3.1.2 Srovnání konfigurací

měřená konfigurace	velikost datové části Echo Request paketu [B] 16	56	120	248	504	1016	2040	4088	8184	průměr
original	1	1	1	1	1	1	1	1	1	1
aslr-randkstack	0,8	0,73	0,83	0,74	0,98	0,93	0,91	1,64	1,02	0,95
aslr-randmmap	1,07	0,98	0,76	0,64	0,82	0,83	0,91	1,68	1,04	0,97
autoconf-performance	0,94	0,71	0,88	0,85	1	0,98	1,09	1,06	0,95	0,94
autoconf-security	1,13	0,9	1,19	1,05	1,11	0,95	1,03	1,04	1,02	1,04
blackhole	0,99	0,83	1,04	1,06	1,11	1,13	1,38	1,07	0,99	1,06
no-simult-connect	0,69	0,55	0,71	0,61	0,78	0,78	0,86	1,66	1,03	0,85
noexec	0,91	0,76	0,79	1,01	1,26	1,07	1,23	1,44	1,08	1,06
noexec-pageexec	0,78	0,63	0,62	0,61	0,84	0,78	0,9	1,64	1,03	0,87
noexec-segmexec	0,83	0,78	0,71	0,68	1,06	0,91	0,9	1,63	1,02	0,94
rap	0,76	0,57	0,63	0,65	0,76	0,78	0,84	1,61	0,99	0,84
rbac	0,97	0,88	0,63	0,6	0,79	0,8	0,89	1,68	1,02	0,91
sanitize	1,02	0,78	0,85	0,89	1,04	0,9	1,02	1,09	1,08	0,96
socket-limit	0,67	0,56	0,66	0,59	0,8	0,78	0,87	1,76	1	0,85
stackleak	0,71	0,56	0,58	0,62	0,76	0,8	0,89	1,65	1,02	0,84
structleak	0,79	0,59	0,68	0,64	0,88	0,81	0,9	1,64	1,01	0,88
udereference	0,78	0,6	0,63	0,64	0,77	0,79	0,86	1,69	1,05	0,86

Tabulka 13.4: Relativní srovnání konfigurací vůči referenční konfiguraci original



Obrázek 13.2: Relativní srovnání konfigurací vůči referenční konfiguraci original

13.3.1.3 Zhodnocení testu

Většina Grsec úprav snižuje odezvu ICMP paketů do velikosti cca 2048 B složeného z následujících komponent:

$$\begin{aligned} \text{ICMP paket} &= \text{IP hlavička} + \text{ICMP hlavička} + \text{ICMP data} \\ 2048 &= 20 + 8 + 2040 \end{aligned}$$

Příkazem `ifconfig` bylo zjištěno, že MTU přímo připojené sítě je 1500. Na obrázku 13.2 patrný zlom v hodnotě 2040 B tak s fragmentací IP paketu nesouvisí. Rozdíly mohou být dány do značné míry různými časy, kdy probíhaly testy jednotlivých konfigurací. Poněkud lepší výsledky autor očekával od konfigurací obsahujících aktivované volby modifikující síťový zásobník. Kupříkladu konfigurace blackhole, která deaktivuje odpovědi z portů na nichž nikdo nenaslouchá (viz kapitola 9.19) se však umístila spíše hůře.

Z testů 13.5.1, 13.5.2 či 13.4.2 vyplývá, že propustnost spoje je dostatečná. Růstová tendence je patrná. Vzhledem k tomu, že se jedná o měření veličiny silně závislé na prostředí sítě, lze rozdíly jednotlivých konfigurací částečně svalit na nestálost testovaného prostředí (NMMP), nicméně hlavní chyba v tomto případě je patrně již v samotném měření a to v nedostatečně velkém statistickém vzorku.

13.4 Testy nástrojem iperf

Iperf je nástroj k měření propustnosti komunikačního kanálu mezi dvěma počítači. Program může běžet buďto v serverovém či v klientském režimu. Během testu komunikují vždy právě dvě instance programu běžící v nestejném režimu. Prostřednictvím TCP či UDP se snaží přenést co největší objem dat. Server naslouchá standardně na všech dostupných adresách na portu 5001. Tabulka 13.5 vysvětluje význam jednotlivých argumentů předávaných nástroji `iperf` v rámci testů uvedených v této kapitole.

Parametr	Význam
`--parallel`	počet paralelně prováděných přenosů (výsledkem je vždy součet jejich propustností)
`--time`	doba trvání testu (v sekundách)
`--reportstyle`	ovlivnění výstupu nástroje `iperf` (typicky je požadován formát CSV pro snadné strojové zpracování)
`--client`	jméno protistrany čili počítače, na kterém běží serverová část nástroje `iperf`
`--udp`	volba přenosového protokolu UDP (namísto výchozího TCP)
`--bandwidth`	omezení generovaného toku (má smysl pouze při použití přenosového protokolu UDP)

Tabulka 13.5: Parametry příkazu `iperf`

13.4.1 Test iperf_local_tcp

Test propustnosti síťového zásobníku testovaného počítače n TCP spojeními zároveň. Měřena je propustnost pouze v jednom směru TCP spoje. Obě zúčastněné strany běží na témže počítači – na xenmv. Jedno měření trvá 10 s.

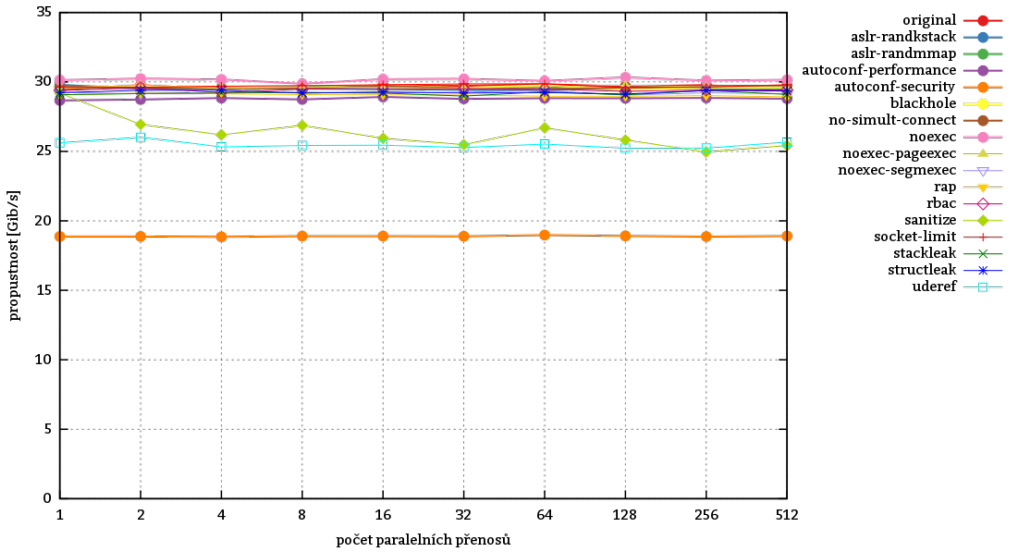
NZPV (x)	počet paralelních přenosů
jednotka NZPV	není (bezrozměrná veličina)
ZPV (y)	propustnost
jednotka ZPV	Gib/s
počet opakování měření	7
počítač spouštějící testovací příkaz	xenmv
testovací příkaz	`iperf --parallel ${X} --time 10 --reportstyle c -w 416K --client ↪ localhost`
počítač spouštějící pomocný příkaz	metalist
pomocný příkaz	`iperf -s`

Tabulka 13.6: Charakteristiky testu iperf_local_tcp

13.4.1.1 Naměřené hodnoty pro jednotlivé konfigurace

měřená konfigurace	počet paralelních přenosů 1	2	4	8	16	32	64	128	256	512
original	29,43	29,76	29,48	29,61	29,74	29,71	29,83	29,67	29,77	29,42
aslr-randkstack	29,79	29,50	29,56	29,52	29,48	29,42	29,38	29,60	29,53	29,36
aslr-randmmap	29,65	29,55	29,56	29,63	29,65	29,55	29,61	29,43	29,62	29,50
autoconf-performance	28,66	28,73	28,84	28,73	28,92	28,78	28,83	28,82	28,85	28,79
autoconf-security	18,87	18,87	18,84	18,90	18,89	18,88	18,97	18,91	18,86	18,90
blackhole	29,55	29,70	29,59	29,57	29,75	29,57	29,75	29,43	29,58	29,58
no-simult-connect	29,63	29,59	29,54	29,52	29,73	29,54	29,47	29,59	29,71	29,72
noexec	30,12	30,23	30,17	29,86	30,19	30,21	30,06	30,32	30,09	30,14
noexec-pageexec	29,58	29,76	29,51	29,64	29,60	29,53	29,67	29,76	29,64	29,64
noexec-segmexec	29,25	29,13	29,28	29,19	29,21	29,26	29,37	29,05	29,21	29,15
rap	29,07	29,19	29,14	29,08	29,06	29,05	28,97	28,97	29,01	28,92
rbac	29,38	29,57	29,24	29,48	29,47	29,50	29,48	29,31	29,41	29,43
sanitize	29,19	26,93	26,19	26,87	25,93	25,48	26,68	25,82	24,95	25,42
socket-limit	29,63	29,61	29,69	29,73	29,79	29,85	29,85	29,60	29,61	29,79
stackleak	29,07	29,16	29,19	29,24	29,19	28,97	29,28	29,06	29,38	29,10
structleak	29,22	29,39	29,40	29,20	29,27	29,20	29,23	29,13	29,37	29,35
uderef	25,62	26,01	25,32	25,42	25,44	25,28	25,51	25,23	25,24	25,66

Tabulka 13.7: Absolutní srovnání konfigurací

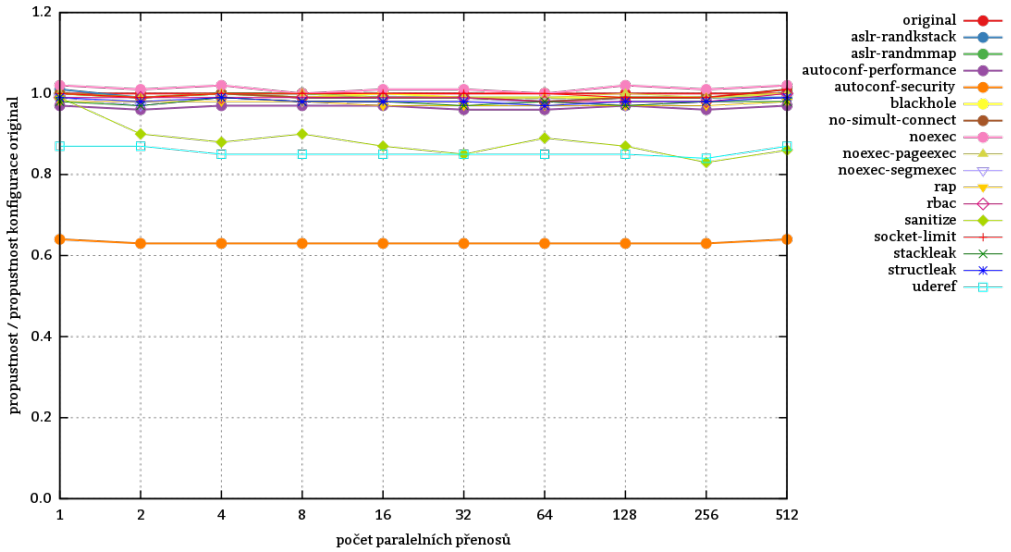


Obrázek 13.3: Absolutní srovnání konfigurací

13.4.1.2 Srovnání konfigurací

měřená konfigurace	počet paralelních přenosů 1	2	4	8	16	32	64	128	256	512	průměr
original	1	1	1	1	1	1	1	1	1	1	1
aslr-randkstack	1,01	0,99	1	0,99	0,99	0,99	0,98	0,99	0,99	0,99	0,99
aslr-randmmap	1	0,99	1	1	0,99	0,99	0,99	0,99	0,99	1	0,99
autoconf-performance	0,97	0,96	0,97	0,97	0,97	0,96	0,96	0,97	0,96	0,97	0,96
autoconf-security	0,64	0,63	0,63	0,63	0,63	0,63	0,63	0,63	0,63	0,64	0,63
blackhole	1	0,99	1	0,99	1	0,99	0,99	0,99	0,99	1	0,99
no-simult-connect	1	0,99	1	0,99	0,99	0,99	0,98	0,99	0,99	1,01	0,99
noexec	1,02	1,01	1,02	1	1,01	1,01	1	1,02	1,01	1,02	1,01
noexec-pageexec	1	0,99	1	1	0,99	0,99	0,99	1	0,99	1	0,99
noexec-segmexec	0,99	0,97	0,99	0,98	0,98	0,98	0,98	0,97	0,98	0,99	0,98
rap	0,98	0,98	0,98	0,98	0,97	0,97	0,97	0,97	0,97	0,98	0,97
rbac	0,99	0,99	0,99	0,99	0,99	0,99	0,98	0,98	0,98	1	0,98
sanitize	0,99	0,9	0,88	0,9	0,87	0,85	0,89	0,87	0,83	0,86	0,88
socket-limit	1	0,99	1	1	1	1	1	0,99	0,99	1,01	0,99
stackleak	0,98	0,97	0,99	0,98	0,98	0,97	0,98	0,97	0,98	0,98	0,97
structleak	0,99	0,98	0,99	0,98	0,98	0,98	0,97	0,98	0,98	0,99	0,98
uderef	0,87	0,87	0,85	0,85	0,85	0,85	0,85	0,85	0,84	0,87	0,85

Tabulka 13.8: Relativní srovnání konfigurací vůči referenční konfiguraci original



Obrázek 13.4: Relativní srovnání konfigurací vůči referenční konfiguraci original

13.4.1.3 Zhodnocení testu

Test je sice prvoplánově cílen na síť, ale **testovaným subjektem se stala hlavní paměť**, jejíž propustnost – až cca 30 Gib/s – se projevila jako úzké hrdlo bránící TCP paketům plynout rychleji. **Propustnost při použití Grsecurity** vyjma konfigurací udereF, sanitize a autoconf-security prakticky **neklesala**. Na síťové prostředí orientovaná konfigurace no-simult-connect test prakticky nemohla ovlivnit. Potlačuje totiž možnost souběžně navazovat spojení a v daném testu se spojení vytváří pouze jednou a pak se již jen používá po celou dobu trvání testu. Dopad na propustnost TCP čili v tomto konkrétním případě na propustnost hlavní paměti test prokázal konfiguracím udereF, sanitize a autoconf-security. Jedinou podstatnou volbou, kterou konfigurace sanitize obsahuje je `PAX_MEMORY_SANITIZE` (viz kapitola 9.13). Fyzické mazání obsahu dealokovaných paměťových stránek způsobuje propad propustnosti hlavní paměti o průměrně 12 %. Volba `PAX_MEMORY_UDEREF` (viz kapitola 9.5.2) zachází s výkonnostní penalizací až na 15 %. No a konečně konfigurace autoconf-security mimo jiné obě tyto volby obsahuje a systém jí postižený je od referenčního systému vzdálen, co se týče výkonnosti, o průměrných 37 %. Test jednoznačně ukázal, že **v systémech, kde je propustnost paměti kritická, není příliš vhodné volit volby** `PAX_MEMORY_SANITIZE` **a** `PAX_MEMORY_UDEREF`. Ostatní volby propustnost paměti nelimitují.

13.4.2 Test iperf_remote_tcp

Test jednosměrné propustnosti proudu dat protokolem TCP. Délka trvání testu je 10 s. Do testu jsou zapojeny dva počítače: xenmv na němž běží serverová instance programu `iperf` a metalist provozující instanci klientskou.

NZPV (x)	počet paralelních přenosů
jednotka NZPV	není (bezrozměrná veličina)
ZPV (y)	propustnost
jednotka ZPV	Gib/s
počet opakování měření	7
počítač spouštějící testovací příkaz	metalist
testovací příkaz	`iperf --parallel ${X} --time 10 --reportstyle c -w 416K --client ↪ xenmv`
počítač spouštějící pomocný příkaz	xenmv
pomocný příkaz	`iperf -s`

Tabulka 13.9: Charakteristiky testu iperf_remote_tcp

13.4.2.1 Naměřené hodnoty pro jednotlivé konfigurace

měřená konfigurace	počet paralelních přenosů 1	2	4	8	16	32	64	128	256	512
original	0,876	0,876	0,876	0,876	0,876	0,876	0,876	0,876	0,876	0,876
aslr-randkstack	0,876	0,876	0,876	0,876	0,876	0,876	0,876	0,876	0,876	0,876
aslr-randmmap	0,876	0,876	0,876	0,876	0,876	0,876	0,876	0,876	0,876	0,876
autoconf-performance	0,876	0,876	0,876	0,876	0,876	0,876	0,876	0,876	0,876	0,876
autoconf-security	0,876	0,876	0,876	0,876	0,876	0,876	0,876	0,876	0,876	0,876
blackhole	0,876	0,876	0,876	0,876	0,876	0,876	0,876	0,876	0,876	0,876
no-simult-connect	0,876	0,876	0,876	0,876	0,876	0,876	0,876	0,876	0,876	0,876
noexec	0,876	0,876	0,876	0,876	0,877	0,876	0,876	0,876	0,876	0,876
noexec-pageexec	0,876	0,876	0,876	0,876	0,876	0,876	0,876	0,876	0,876	0,876
noexec-segmexec	0,876	0,876	0,876	0,876	0,876	0,876	0,876	0,876	0,876	0,876
rap	0,878	0,878	0,878	0,878	0,878	0,878	0,878	0,878	0,878	0,878
rbac	0,876	0,876	0,876	0,876	0,876	0,876	0,876	0,876	0,876	0,876
sanitize	0,878	0,878	0,878	0,878	0,878	0,878	0,878	0,878	0,878	0,878
socket-limit	0,876	0,876	0,876	0,876	0,876	0,876	0,876	0,876	0,876	0,876
stackleak	0,876	0,876	0,876	0,876	0,876	0,876	0,876	0,876	0,877	0,876
structleak	0,876	0,876	0,876	0,876	0,876	0,876	0,876	0,876	0,876	0,876
uderef	0,876	0,876	0,876	0,876	0,876	0,876	0,876	0,876	0,876	0,876

Tabulka 13.10: Absolutní srovnání konfigurací

13.4.2.2 Srovnání konfigurací

měřená konfigurace	počet paralelních přenosů 1	2	4	8	16	32	64	128	256	512	průměr
original	1	1	1	1	1	1	1	1	1	1	1
aslr-randkstack	1	1	1	1	1	1	1	1	1	1	1
aslr-randmmap	1	1	1	1	1	1	1	1	1	1	1
autoconf-performance	1	1	1	1	1	1	1	1	1	1	1
autoconf-security	1	1	1	1	1	1	1	1	1	1	1
blackhole	1	1	1	1	1	1	1	1	1	1	1
no-simult-connect	1	1	1	1	1	1	1	1	1	1	1
noexec	1	1	1	1	1	1	1	1	1	1	1
noexec-pageexec	1	1	1	1	1	1	1	1	1	1	1
noexec-segmexec	1	1	1	1	1	1	1	1	1	1	1
rap	1	1	1	1	1	1	1	1	1	1	1
rbac	1	1	1	1	1	1	1	1	1	1	1
sanitize	1	1	1	1	1	1	1	1	1	1	1
socket-limit	1	1	1	1	1	1	1	1	1	1	1
stackleak	1	1	1	1	1	1	1	1	1	1	1
structleak	1	1	1	1	1	1	1	1	1	1	1
uderef	1	1	1	1	1	1	1	1	1	1	1

Tabulka 13.11: Relativní srovnání konfigurací vůči referenční konfiguraci original

13.4.2.3 Zhodnocení testu

Všechny konfigurace v testu dopadly podobně a to natolik, že v tabulce 13.10 díky zaokrouhlení není zřetelný naprosto žádný rozdíl. Na rozdíl od testu v kapitole 13.4.1 zde jako úzké hrdlo nefiguruje hlavní paměť, nýbrž nejslabší článek spoje (viz kapitola 11.1), který má přenosovou kapacitu natolik nízkou, že **nároky kladené na výkonnost systému byly dostatečně malé na to, aby nebyl nikdy svou konfigurací limitován**. Přenosová rychlost 0,876 Gib/s čili 0,94 Gb/s koresponduje i s maximální teoretickou propustností spoje 1 Gb/s dle kapitoly 11.1.

13.5 Testy nástrojem qperf

`qperf` je nástroj pro zjišťování parametrů síťových spojů, který v době měření nebyl součástí standardního repozitáře pro Debian 8. `qperf` k měření vyžaduje vždy běh dvou svých instancí. Měřeno je vždy jednosměrné zpoždění či jednosměrná propustnost vždy ve směru klient → server.

13.5.1 Test qperf_local_bw_tcp

Test `qperf_local_bw_tcp` zjišťuje jednosměrnou propustnost síťového subsystému při použití protokolu TCP. Testu se účastní dvě instance programu `qperf` – jedna v roli serveru přijímajícího data a druhá v roli klienta. Test trvá 10 s.

měřená veličina	propustnost
jednotka měřené veličiny	Gb/s
počet opakování měření	7
počítač spouštějící testovací příkaz	xenmv
testovací příkaz	`qperf 127.0.0.1 --precision 10 --time 10 --unify_units ↪ --use_bits_per_sec tcp_bw`
počítač spouštějící pomocný příkaz	metalist
pomocný příkaz	`qperf`

Tabulka 13.12: Charakteristiky testu qperf_local_bw_tcp

13.5.1.1 Naměřené hodnoty pro jednotlivé konfigurace

měřená konfigurace	propustnost [Gb/s]
original	29,20
aslr-randkstack	31,37
aslr-randmmap	31,33
autoconf-performance	29,79
autoconf-security	20,27
blackhole	32,02
no-simult-connect	32,24
noexec	32,24
noexec-pageexec	30,51
noexec-segmexec	31,96
rap	30,80
rbac	31,24
sanitize	30,51
socket-limit	32,14
stackleak	30,88
structleak	30,93
uderef	26,29

Tabulka 13.13: Absolutní srovnání konfigurací

13.5.1.2 Srovnání konfigurací

měřená konfigurace	propustnost [Gb/s]
original	1
aslr-randkstack	1,07
aslr-randmmap	1,07
autoconf-performance	1,01
autoconf-security	0,69
blackhole	1,09
no-simult-connect	1,1
noexec	1,1
noexec-pageexec	1,04
noexec-segmexec	1,09
rap	1,05
rbac	1,06
sanitize	1,04
socket-limit	1,1
stackleak	1,05
structleak	1,05
uderef	0,9

Tabulka 13.14: Relativní srovnání konfigurací vůči referenční konfiguraci original

13.5.1.3 Zhodnocení testu

Propustnost cca 30 Gb/s představuje stejně tak, jako i v testu uvedeném v kapitole 13.4.1, propustnost hlavní paměti, neboť právě ta je úzkým hrdlem v případě komunikace zasílané TCP proudem na virtuální síťové rozhraní „localhost“. V některých konfiguracích je pozorovatelné dokonce mírné zlepšení oproti referenční konfiguraci, nicméně to bude velmi pravděpodobně způsobeno NMMP. Výrazněji zaostávají pouze konfigurace uderef a autoconf-security. Vztah k. uderef vzhledem ke k. autoconf-security byl již popsán v kapitole 13.4.1 a proto již není překvapením, proč dané konfigurace dopadly, jak dopadly. Pro čtenáře obeznámeného s výsledky dosaženými v testu uvedeném v kapitole 13.4.1 je však překvapivé jistě to, že konfigurace sanitize s sebou do systému nepřinesla žádnou výkonnostní ztrátu, jak tomu bylo v testu 13.4.1. Jak je to možné? Testovací nástroj `qperf` se patrně v takové míře nezbavuje již jednou mu přidělené paměti. Buďto ji používá opakovaně, nebo si ji prostě jen drží a nedealokuje ji. Jádru tak nepřidělává práci s jejím „nulováním“ a hlavní paměť se tak může plně věnovat čtení a zápisu přenášených dat. Oddělení paměťových prostorů Kernel Space a User Space v podání konfigurace uderef však testovací nástroj obejít nedokázal. Po celou dobu běžel v user mode, čili využíval User Space, a síťový zásobník v držení jádra a tudíž i kopírování dat mezi User Space a Kernel Space si vybrali svou daň v podobě snížení propustnosti hlavní paměti o 10 %. **Chce-li čtenář nasadit Grsecurity na systému, kde propustnost paměti hraje prim, tak nechť raději dvakrát rozmyslí aktivaci voleb** `PAX_MEMORY_UDEREF` **a** `GRKERNSEC_CONFIG_PRIORITY_SECURITY`.

13.5.2 Test qperf_local_bw_udp

Test `qperf_local_bw_udp` je obdobou testu uvedeného v kapitole 13.5.1. Jedinou odlišností je protokol použitý pro přenos dat. V případě tohoto testu je jím UDP.

13.5.2.1 Naměřené hodnoty pro jednotlivé konfigurace

měřená veličina	propustnost
jednotka měřené veličiny	Gb/s
počet opakování měření	7
počítač spouštějící testovací příkaz	xenmv
testovací příkaz	`qperf 127.0.0.1 --precision 10 --time 10 --unify_units ↪ --use_bits_per_sec udp_bw`
počítač spouštějící pomocný příkaz	metalist
pomocný příkaz	`qperf`

Tabulka 13.15: Charakteristiky testu qperf_local_bw_udp

měřená konfigurace	**propustnost [Gb/s]**
original	33,70
aslr-randkstack	34,98
aslr-randmmap	34,76
autoconf-performance	36,84
autoconf-security	22,39
blackhole	32,48
no-simult-connect	32,73
noexec	34,98
noexec-pageexec	33,18
noexec-segmexec	38,13
rap	34,27
rbac	34,10
sanitize	34,94
socket-limit	33,38
stackleak	36,40
structleak	33,35
uderef	34,46

Tabulka 13.16: Absolutní srovnání konfigurací

13.5.2.2 Srovnání konfigurací

měřená konfigurace	propustnost [Gb/s]
original	1
aslr-randkstack	1,03
aslr-randmmap	1,03
autoconf-performance	1,09
autoconf-security	0,66
blackhole	0,96
no-simult-connect	0,97
noexec	1,03
noexec-pageexec	0,98
noexec-segmexec	1,13
rap	1,01
rbac	1,01
sanitize	1,03
socket-limit	0,99
stackleak	1,08
structleak	0,98
uderef	1,02

Tabulka 13.17: Relativní srovnání konfigurací vůči referenční konfiguraci original

13.5.2.3 Zhodnocení testu

Propustnost cca 34 Gb/s představuje propustnost hlavní paměti, neboť právě ta je úzkým hrdlem v případě komunikace zasílané UDP proudem na virtuální síťové rozhraní „localhost“. Za povšimnutí rozhodně stojí fakt, že propustnost hlavní paměti naměřená UDP proudem je o cca 13 % vyšší nežli její propustnost naměřená TCP spojením. Konfiguraci uderef test neprokázal zpomalení systému, jako tomu bylo v předcházejícím testu. Zajímavého výkonnostního posunu – +13 % – bylo dosaženo s konfigurací noexec-segmexec. Vzhledem k počtu opakování testu by NMMP 13 % rozdílu udělat nemělo, nicméně autor tento stav neumí logicky vysvětlit. Nějaké zvláštní doporučení však konfiguraci noexec-segmexec neudělí, neboť ta byla do měření zahrnuta výhradně pro zjištění dopadu emulace NX bitu a nasazování x86 systémů bez podpory 64 bitových instrukcí se příliš nepředpokládá.

13.5.3 Test qperf_local_lat_tcp

Test `qperf_local_lat_tcp` zjišťuje jednosměrné zpoždění paketů od jejich odeslání jednou instancí programu `qperf` až do přijetí druhou instancí programu `qperf`. Měřené zpoždění tak zahrnuje i prodlevy v rámci síťového zásobníku. Pro přenos je využit protokol TCP. Vše probíhá v rámci počítače xenmv.

měřená veličina	zpoždění
jednotka měřené veličiny	µs
počet opakování měření	7
počítač spouštějící testovací příkaz	xenmv
testovací příkaz	`qperf 127.0.0.1 --precision 10 --time 10 --unify_units --use_bits_per_sec tcp_lat`
počítač spouštějící pomocný příkaz	metalist
pomocný příkaz	`qperf`

Tabulka 13.18: Charakteristiky testu qperf_local_lat_tcp

13.5.3.1 Naměřené hodnoty pro jednotlivé konfigurace

měřená konfigurace	zpoždění [µs]
original	10,105
aslr-randkstack	10,549
aslr-randmmap	10,759
autoconf-performance	10,708
autoconf-security	11,443
blackhole	10,412
no-simult-connect	10,423
noexec	10,127
noexec-pageexec	10,847
noexec-segmexec	10,815
rap	11,137
rbac	10,989
sanitize	10,571
socket-limit	10,397
stackleak	11,369
structleak	10,723
uderef	12,14

Tabulka 13.19: Absolutní srovnání konfigurací

13.5.3.2 Srovnání konfigurací

měřená konfigurace	zpoždění [µs]
original	1
aslr-randkstack	1,04
aslr-randmmap	1,06
autoconf-performance	1,05
autoconf-security	1,13
blackhole	1,03
no-simult-connect	1,03
noexec	1
noexec-pageexec	1,07
noexec-segmexec	1,07
rap	1,1
rbac	1,08
sanitize	1,04
socket-limit	1,02
stackleak	1,12
structleak	1,06
uderef	1,2

Tabulka 13.20: Relativní srovnání konfigurací vůči referenční konfiguraci original

13.5.3.3 Zhodnocení testu

V jednosměrném zpoždění při komunikaci protokolem TCP se od referenční konfigurace výrazněji odchýlily pouze konfigurace uderef (pokles o 22 %), autoconf-security (pokles o 19 %) a stackleak (pokles o 13 %). Oddělení paměťových prostorů jednotlivých ringů zapříčiněné k. uderef, zdá se, vyžaduje určitý nezanedbatelný čas navíc k přenosu informace z Kernel Space do User Space. Zároveň čištění kernelspace-stack při každém návratu ze systémového volání (konfigurace stackleak) také brzdí přenos TCP paketů. Ostatní konfigurace neměly na odezvu TCP paketů dopad a jejich rozptyl lze přičíst na vrub NMMP.

13.5.4 Test qperf_local_lat_udp

Test `qperf_local_lat_udp` zjišťuje jednosměrné zpoždění paketů čili dobu od jejich odeslání jednou instancí programu `qperf` až do přijetí druhou instancí programu `qperf`. Měřené zpoždění tak zahrnuje i prodlevy v rámci síťového zásobníku. Pro přenos je využit protokol UDP. Test trvá 10 s.

měřená veličina	zpoždění
jednotka měřené veličiny	µs
počet opakování měření	7
počítač spouštějící testovací příkaz	xenmv
testovací příkaz	`qperf 127.0.0.1 --precision 10 --time 10 --unify_units ↪ --use_bits_per_sec udp_lat`
počítač spouštějící pomocný příkaz	metalist
pomocný příkaz	`qperf`

Tabulka 13.21: Charakteristiky testu qperf_local_lat_udp

13.5.4.1 Naměřené hodnoty pro jednotlivé konfigurace

měřená konfigurace	zpoždění [µs]
original	7,951
aslr-randkstack	8,307
aslr-randmmap	8,522
autoconf-performance	8,634
autoconf-security	9,536
blackhole	8,828
no-simult-connect	8,586
noexec	8,37
noexec-pageexec	9,009
noexec-segmexec	8,933
rap	8,692
rbac	8,597
sanitize	8,599
socket-limit	8,647
stackleak	9,026
structleak	8,895
uderef	9,734

Tabulka 13.22: Absolutní srovnání konfigurací

13.5.4.2 Srovnání konfigurací

měřená konfigurace	zpoždění [µs]
original	1
aslr-randkstack	1,04
aslr-randmmap	1,07
autoconf-performance	1,08
autoconf-security	1,19
blackhole	1,11
no-simult-connect	1,07
noexec	1,05
noexec-pageexec	1,13
noexec-segmexec	1,12
rap	1,09
rbac	1,08
sanitize	1,08
socket-limit	1,08
stackleak	1,13
structleak	1,11
uderef	1,22

Tabulka 13.23: Relativní srovnání konfigurací vůči referenční konfiguraci original

13.5.4.3 Zhodnocení testu

Test ukázal obdobné výsledky jako test uvedený v kapitole 13.5.3. Zde je zajímavé si všimnout též rozdílu ve zpoždění přenosu TCP paketu a UDP paketu. Ten činí cca 2,3 µs ve prospěch UDP. V testu nejvíce zaostávají tak jako i dříve systémy s konfiguracemi uderef a autoconf-security.

13.5.5 Test qperf_remote_bw_tcp

Test `qperf_remote_bw_tcp` zjišťuje jednosměrnou propustnost síťového subsystému při použití protokolu TCP. Testu se účastní dvě instance programu `qperf` – jedna v roli serveru přijímajícího data a druhá v roli klienta. Test trvá 10 s. Klientská instance programu `qperf` běží na jiném počítači nežli její protějšek.

měřená veličina	propustnost
jednotka měřené veličiny	Gb/s
počet opakování měření	7
počítač spouštějící testovací příkaz	metalist
testovací příkaz	`qperf xenmv --precision 10 --time 10 --unify_units --use_bits_per_sec tcp_bw`
počítač spouštějící pomocný příkaz	xenmv
pomocný příkaz	`qperf`

Tabulka 13.24: Charakteristiky testu qperf_remote_bw_tcp

13.5.5.1 Naměřené hodnoty pro jednotlivé konfigurace

měřená konfigurace	propustnost [Gb/s]
original	0,941
aslr-randkstack	0,941
aslr-randmmap	0,941
autoconf-performance	0,94
autoconf-security	0,941
blackhole	0,941
no-simult-connect	0,941
noexec	0,941
noexec-pageexec	0,94
noexec-segmexec	0,941
rap	0,941
rbac	0,941
sanitize	0,941
socket-limit	0,941
stackleak	0,941
structleak	0,941
uderef	0,941

Tabulka 13.25: Absolutní srovnání konfigurací

13.5.5.2 Srovnání konfigurací

měřená konfigurace	propustnost [Gb/s]
original	1
aslr-randkstack	1
aslr-randmmap	1
autoconf-performance	0,99
autoconf-security	1
blackhole	1
no-simult-connect	1
noexec	1
noexec-pageexec	0,99
noexec-segmexec	1
rap	1
rbac	1
sanitize	1
socket-limit	1
stackleak	1
structleak	1
uderef	1

Tabulka 13.26: Relativní srovnání konfigurací vůči referenční konfiguraci original

13.5.5.3 Zhodnocení testu

Test prokázal, že jedním TCP spojením využitelná přenosová kapacita spoje mezi počítači xenmv a metalist je nanejvýš 0,941 Gb/s. Žádná konfigurace zjišťovanou veličinu nijak nepoznamenala. Úzkým hrdlem je v tomto případě nejpomalejší prvek na spoji (viz kapitola 11.1).

13.5.6 Test qperf_remote_bw_udp

Test `qperf_remote_bw_udp` je obdobou testu uvedeného v kapitole 13.5.1. Jedinou odlišností je protokol použitý pro přenos dat. V případě tohoto testu je jím UDP.

13.5.6.1 Naměřené hodnoty pro jednotlivé konfigurace

měřená veličina	propustnost
jednotka měřené veličiny	Gb/s
počet opakování měření	7
počítač spouštějící testovací příkaz	metalist
testovací příkaz	`qperf xenmv --precision 10 --time 10 --unify_units ↪ --use_bits_per_sec udp_bw`
počítač spouštějící pomocný příkaz	xenmv
pomocný příkaz	`qperf`

Tabulka 13.27: Charakteristiky testu qperf_remote_bw_udp

měřená konfigurace	**propustnost [Gb/s]**
original	0,958
aslr-randkstack	0,959
aslr-randmmap	0,96
autoconf-performance	0,96
autoconf-security	0,959
blackhole	0,96
no-simult-connect	0,96
noexec	0,959
noexec-pageexec	0,96
noexec-segmexec	0,959
rap	0,958
rbac	0,959
sanitize	0,96
socket-limit	0,96
stackleak	0,96
structleak	0,96
uderef	0,96

Tabulka 13.28: Absolutní srovnání konfigurací

13.5.6.2 Srovnání konfigurací

měřená konfigurace	propustnost [Gb/s]
original	1
aslr-randkstack	1
aslr-randmmap	1
autoconf-performance	1
autoconf-security	1
blackhole	1
no-simult-connect	1
noexec	1
noexec-pageexec	1
noexec-segmexec	1
rap	1
rbac	1
sanitize	1
socket-limit	1
stackleak	1
structleak	1
uderef	1

Tabulka 13.29: Relativní srovnání konfigurací vůči referenční konfiguraci original

13.5.6.3 Zhodnocení testu

Test prokázal, že jedním UDP spojením využitelná přenosová kapacita spoje mezi počítači xenmv a metalist je nanejvýš 0,96 Gb/s[1]. Žádná konfigurace zjišťovanou veličinu nijak nepoznamenala. Úzkým hrdlem je tak i v tomto případě použitý hardware (viz kapitola 11.1).

13.5.7 Test qperf_remote_lat_tcp

Test `qperf_remote_lat_tcp` zjišťuje jednosměrné zpoždění paketů od jejich odeslání jednou instancí programu `qperf` až do přijetí druhou instancí programu `qperf`. Měřené zpoždění tak zahrnuje i prodlevy v rámci síťových zásobníků strojů účastnících se měření – xenmv a metalist. Pro přenos je využit protokol TCP.

měřená veličina	zpoždění
jednotka měřené veličiny	µs
počet opakování měření	7
počítač spouštějící testovací příkaz	metalist
testovací příkaz	`qperf xenmv --precision 10 --time 10 --unify_units --use_bits_per_sec tcp_lat`
počítač spouštějící pomocný příkaz	xenmv
pomocný příkaz	`qperf`

Tabulka 13.30: Charakteristiky testu qperf_remote_lat_tcp

13.5.7.1 Naměřené hodnoty pro jednotlivé konfigurace

[1] 2

měřená konfigurace	zpoždění [µs]
original	49,93
aslr-randkstack	49,92
aslr-randmmap	49,92
autoconf-performance	49,92
autoconf-security	49,92
blackhole	49,93
no-simult-connect	49,92
noexec	49,92
noexec-pageexec	49,92
noexec-segmexec	49,92
rap	49,93
rbac	49,92
sanitize	49,92
socket-limit	49,98
stackleak	49,92
structleak	50,40
uderef	49,95

Tabulka 13.31: Absolutní srovnání konfigurací

13.5.7.2 Srovnání konfigurací

měřená konfigurace	zpoždění [µs]
original	1
aslr-randkstack	0,99
aslr-randmmap	0,99
autoconf-performance	0,99
autoconf-security	0,99
blackhole	0,99
no-simult-connect	0,99
noexec	0,99
noexec-pageexec	0,99
noexec-segmexec	0,99
rap	0,99
rbac	0,99
sanitize	0,99
socket-limit	1
stackleak	0,99
structleak	1
uderef	1

Tabulka 13.32: Relativní srovnání konfigurací vůči referenční konfiguraci original

13.5.7.3 Zhodnocení testu

Test neprokázal sebemenší vliv jednotlivých konfigurací systému na jednosměrné zpoždění TCP paketů. Poslouží tedy alespoň pro srovnání jednosměrných zpoždění:

- ~ 10,5 µs pro cestu: xenmv $\xrightarrow{\text{TCP}}$ xenmv
- ~ 49,9 µs pro cestu: metalist $\xrightarrow{\text{TCP}}$ přepínač $\xrightarrow{\text{TCP}}$ xenmv

Zpoždění paketu, který neopustí xenmv a projde jen jeho loopbackem je 5 krát menší nežli zpoždění paketu jdoucího skrze přepínač do metalist. Zpoždění způsobené síťovým prvkem je tedy cca 40 µs.

13.5.8 Test qperf_remote_lat_udp

Test `qperf_remote_lat_udp` zjišťuje jednosměrné zpoždění paketů od jejich odeslání jednou instancí programu `qperf` až do přijetí druhou instancí programu `qperf`. Měřené zpoždění tak zahrnuje i prodlevy v rámci síťového zásobníku. Pro přenos je využit protokol UDP.

měřená veličina	zpoždění
jednotka měřené veličiny	µs
počet opakování měření	7
počítač spouštějící testovací příkaz	metalist
testovací příkaz	`qperf xenmv --precision 10 --time 10 --unify_units ↪ --use_bits_per_sec udp_lat`
počítač spouštějící pomocný příkaz	xenmv
pomocný příkaz	`qperf`

Tabulka 13.33: Charakteristiky testu qperf_remote_lat_udp

13.5.8.1 Naměřené hodnoty pro jednotlivé konfigurace

měřená konfigurace	zpoždění [µs]
original	49,915
aslr-randkstack	49,918
aslr-randmmap	49,911
autoconf-performance	49,878
autoconf-security	49,899
blackhole	49,918
no-simult-connect	49,9
noexec	49,912
noexec-pageexec	49,924
noexec-segmexec	49,919
rap	49,919
rbac	49,918
sanitize	49,991
socket-limit	49,965
stackleak	49,932
structleak	49,92
uderef	49,971

Tabulka 13.34: Absolutní srovnání konfigurací

13.5.8.2 Srovnání konfigurací

měřená konfigurace	zpoždění [µs]
original	1
aslr-randkstack	1
aslr-randmmap	0,99
autoconf-performance	0,99
autoconf-security	0,99
blackhole	1
no-simult-connect	0,99
noexec	0,99
noexec-pageexec	1
noexec-segmexec	1
rap	1
rbac	1
sanitize	1
socket-limit	1
stackleak	1
structleak	1
uderef	1

Tabulka 13.35: Relativní srovnání konfigurací vůči referenční konfiguraci original

13.5.8.3 Zhodnocení testu

Test neprokázal sebemenší vliv jednotlivých konfigurací systému na jednosměrné zpoždění UDP paketů.

13.6 Testy nástrojem lmbench

Lmbench je multiplatformí benchmark psaný v jazyce ANSI C. Obsahuje řadu testů, z nichž některé byly použity v rámci této práce. V systému Debian 8.6 standardně přítomen není a bylo jej proto potřeba získat z oficiálních stránek testu `http://www.bitmover.com/lmbench/`. Jedním z parametrů, které přebírá je počet procesů provádějících test. V případě více do testu zapojených procesů každý proces představuje instanci téhož programu a zjišťuje tutéž NZPV [Sta02]. Každý proces měřením dojde k určité hodnotě. Výslednou hodnotu pak představuje průměr ze všech hodnot naměřených jednotlivými procesy. Účelem parametrizace testu počtem procesů je zjištění, jak daný systém škáluje se zvyšující se zátěží.

13.6.1 Test lmbench_lat_connect_local

Program `lat_connect` měří dobu potřebnou pro navázání spojení mezi dvěma procesy běžícími na témže počítači prostřednictvím AF_INET socketu. Měřená doba zahrnuje čas, který uplyne od iniciování TCP spojení na klientské straně do přechodu spojení do stavu Established u obou komunikujících stran [SM94]. Test nezahrnuje čas strávený překladem jména.

měřená veličina	čas potřebný pro navázání spojení prostřednictvím TCP socketu
jednotka měřené veličiny	µs
počet opakování měření	21
počítač spouštějící testovací příkaz	xenmv
testovací příkaz	`lat_connect -N 5 localhost 2>&1 | tee /dev/stderr`
počítač spouštějící pomocný příkaz	metalist
pomocný příkaz	`lat_connect -s`

Tabulka 13.36: Charakteristiky testu lmbench_lat_connect_local

13.6.1.1 Naměřené hodnoty pro jednotlivé konfigurace

měřená konfigurace	čas potřebný pro navázání spojení prostřednictvím TCP socketu [µs]
original	34,32
aslr-randkstack	33,53
aslr-randmmap	35,66
autoconf-performance	35,46
autoconf-security	35,94
blackhole	32,63
no-simult-connect	24,75
noexec	32,47
noexec-pageexec	34,07
noexec-segmexec	34,50
rap	34,58
rbac	33,39
sanitize	33,41
socket-limit	33,00
stackleak	41,48
structleak	33,16
uderef	35,71

Tabulka 13.37: Absolutní srovnání konfigurací

13.6.1.2 Srovnání konfigurací

měřená konfigurace	čas potřebný pro navázání spojení prostřednictvím TCP socketu [µs]
original	1
aslr-randkstack	0,97
aslr-randmmap	1,03
autoconf-performance	1,03
autoconf-security	1,04
blackhole	0,95
no-simult-connect	0,72
noexec	0,94
noexec-pageexec	0,99
noexec-segmexec	1
rap	1
rbac	0,97
sanitize	0,97
socket-limit	0,96
stackleak	1,2
structleak	0,96
uderef	1,04

Tabulka 13.38: Relativní srovnání konfigurací vůči referenční konfiguraci original

13.6.1.3 Zhodnocení testu

Měření ukázalo, že Grsec navazování spojení skrz AF_INET sockety nijak nedegraduje vyjma konfigurace stackleak. Navázání spojení probíhá nejrychleji na konfiguraci no-simult-connect – odstranění podpory souběžného spojení z TCP (viz kapitola 9.19) zde přineslo vedlejší efekt v podobě zkrácení doby potřebné pro navázání spojení o 28 %.

13.6.2 Test lmbench_lat_connect_remote

Program `lat_connect` měří dobu potřebnou pro navázání spojení mezi dvěma procesy běžícími na **dvou různých počítačích** prostřednictvím AF_INET socketu. Měřená doba zahrnuje čas, který uplyne od iniciování TCP spojení na klientské straně do přechodu spojení do stavu Established u obou komunikujících stran [SM94]. Test nezahrnuje čas strávený překladem jména.

měřená veličina	čas potřebný pro navázání spojení prostřednictvím TCP socketu
jednotka měřené veličiny	µs
počet opakování měření	21
počítač spouštějící testovací příkaz	metalist
testovací příkaz	`lat_connect -N 5 xenmv 2>&1 | tee /dev/stderr`
počítač spouštějící pomocný příkaz	xenmv
pomocný příkaz	`lat_connect -s`

Tabulka 13.39: Charakteristiky testu lmbench_lat_connect_remote

13.6.2.1 Naměřené hodnoty pro jednotlivé konfigurace

měřená konfigurace	čas potřebný pro navázání spojení prostřednictvím TCP socketu [µs]
original	2148,69
aslr-randkstack	2146,96
aslr-randmmap	2140,35
autoconf-performance	2137,21
autoconf-security	2139,81
blackhole	2140,91
no-simult-connect	2141,79
noexec	2148,45
noexec-pageexec	2148,95
noexec-segmexec	2135,68
rap	2135,44
rbac	2129,35
sanitize	2147,04
socket-limit	2144,20
stackleak	2149,57
structleak	2150,50
uderef	2162,56

Tabulka 13.40: Absolutní srovnání konfigurací

13.6.2.2 Srovnání konfigurací

měřená konfigurace	čas potřebný pro navázání spojení prostřednictvím TCP socketu [µs]
original	1
aslr-randkstack	0,99
aslr-randmmap	0,99
autoconf-performance	0,99
autoconf-security	0,99
blackhole	0,99
no-simult-connect	0,99
noexec	0,99
noexec-pageexec	1
noexec-segmexec	0,99
rap	0,99
rbac	0,99
sanitize	0,99
socket-limit	0,99
stackleak	1
structleak	1
uderef	1

Tabulka 13.41: Relativní srovnání konfigurací vůči referenční konfiguraci original

13.6.2.3 Zhodnocení testu

Test nezjistil rozdílu mezi jednotlivými konfiguracemi. Naměřené absolutní hodnoty jsou o dva řády vyšší nežli v testu kapitoly 13.6.1, kde se nějaké rozdíly mezi konfiguracemi projevily.

13.6.3 Test lmbench_lat_unix_connect

Měření `lmbench_lat_unix_connect` měří čas strávený navazováním spojení skrze socket rodiny AF_UNIX. V rámci jednoho testu je spuštěna dvojice procesů. Měří se doba vytvoření AF_UNIX socketu včetně navázání spojení s protistranou. Příkaz iniciující testování upravuje pouze počet dvojic procesů, které budou komunikovat „zároveň“ – NZPV v rámci daného měření. Socket je během testování mapován na dočasný soubor ležící v adreáři `/tmp`.

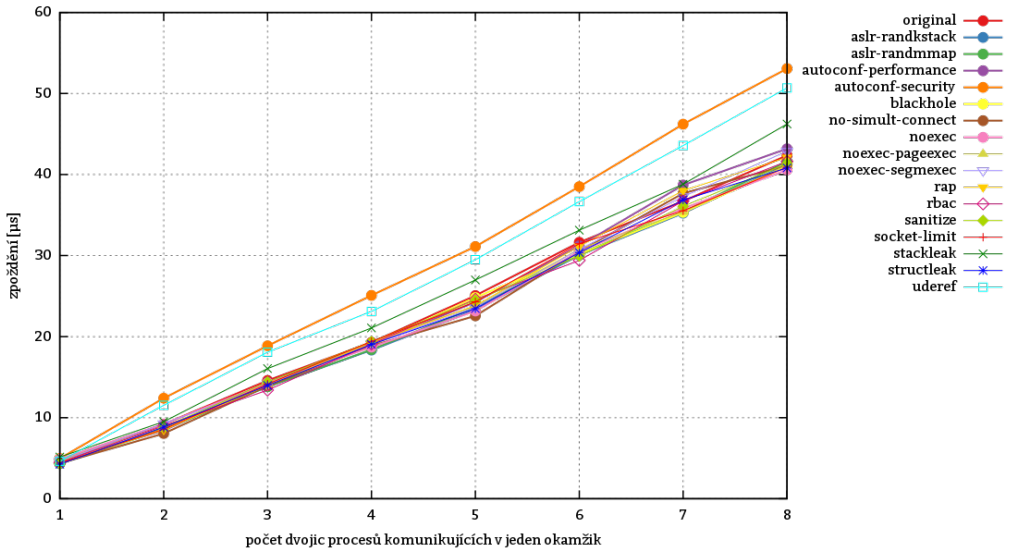
NZPV (x)	počet dvojic procesů komunikujících v jeden okamžik
jednotka NZPV	není (bezrozměrná veličina)
ZPV (y)	zpoždění
jednotka ZPV	µs
počet opakování měření	7
počítač spouštějící testovací příkaz	xenmv
testovací příkaz	`lat_unix_connect -P ${X} -N 5 2>&1 | tee /dev/stderr`
počítač spouštějící pomocný příkaz	metalist
pomocný příkaz	`lat_unix_connect -s`

Tabulka 13.42: Charakteristiky testu lmbench_lat_unix_connect

13.6.3.1 Naměřené hodnoty pro jednotlivé konfigurace

měřená konfigurace	počet dvojic procesů komunikujících v jeden okamžik 1	2	3	4	5	6	7	8
original	4,655	9,18	14,584	19,28	25,06	31,63	36,71	42,36
aslr-randkstack	4,321	8,72	13,837	18,33	23,31	30,06	35,23	41,24
aslr-randmmap	4,388	8,50	13,819	18,39	23,62	30,23	35,40	41,61
autoconf-performance	4,744	8,41	14,367	19,00	24,26	31,20	38,69	43,17
autoconf-security	4,97	12,40	18,87	25,08	31,11	38,51	46,21	53,07
blackhole	4,44	8,53	14,211	18,95	24,81	30,19	35,32	40,99
no-simult-connect	4,37	8,04	13,922	18,86	22,56	30,52	37,69	41,12
noexec	4,885	9,23	14,276	18,70	23,16	30,51	35,82	40,57
noexec-pageexec	4,727	8,94	14,104	19,35	23,79	30,80	35,38	41,15
noexec-segmexec	4,61	9,24	14,044	19,11	23,62	30,75	37,29	42,87
rap	4,567	8,49	13,978	19,20	23,83	31,15	38,01	42,10
rbac	4,457	8,99	13,403	19,08	24,62	29,39	36,74	41,61
sanitize	4,288	8,77	14,405	19,40	24,48	29,94	36,07	41,33
socket-limit	4,389	8,58	14,151	19,41	24,29	31,46	35,53	40,91
stackleak	5,151	9,50	16,061	21,07	26,99	33,13	38,81	46,23
structleak	4,252	8,83	13,986	19,03	23,45	30,37	36,86	40,81
uderef	4,628	11,53	18,084	23,11	29,49	36,65	43,57	50,68

Tabulka 13.43: Absolutní srovnání konfigurací

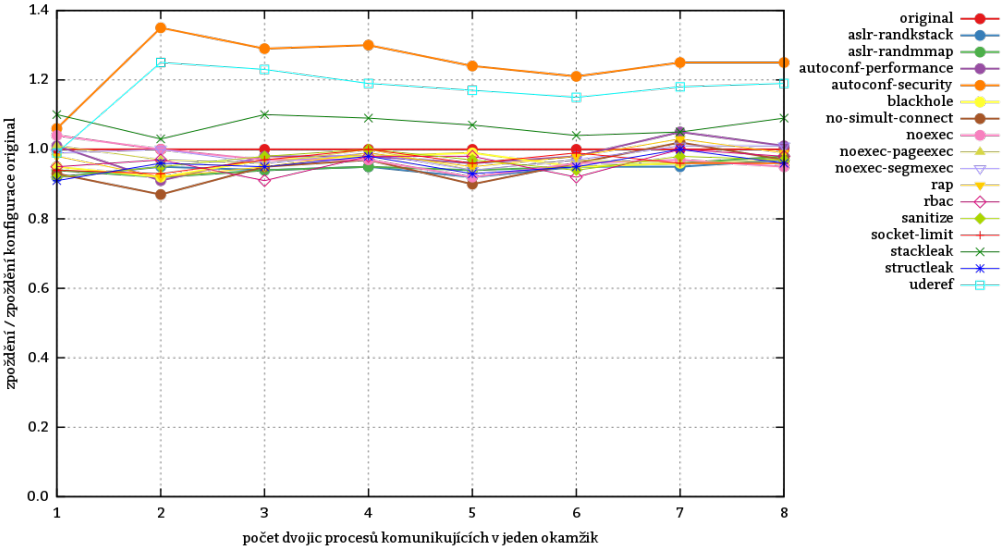


Obrázek 13.5: Absolutní srovnání konfigurací

13.6.3.2 Srovnání konfigurací

měřená konfigurace	počet dvojic procesů komunikujících v jeden okamžik 1	2	3	4	5	6	7	8	průměr
original	1	1	1	1	1	1	1	1	1
aslr-randkstack	0,92	0,95	0,94	0,95	0,92	0,95	0,95	0,97	0,94
aslr-randmmap	0,94	0,92	0,94	0,95	0,94	0,95	0,96	0,98	0,94
autoconf-performance	1,01	0,91	0,98	0,98	0,96	0,98	1,05	1,01	0,98
autoconf-security	1,06	1,35	1,29	1,3	1,24	1,21	1,25	1,25	1,24
blackhole	0,95	0,92	0,97	0,98	0,99	0,95	0,96	0,96	0,96
no-simult-connect	0,93	0,87	0,95	0,97	0,9	0,96	1,02	0,97	0,94
noexec	1,04	1	0,97	0,97	0,92	0,96	0,97	0,95	0,97
noexec-pageexec	1,01	0,97	0,96	1	0,94	0,97	0,96	0,97	0,97
noexec-segmexec	0,99	1	0,96	0,99	0,94	0,97	1,01	1,01	0,98
rap	0,98	0,92	0,95	0,99	0,95	0,98	1,03	0,99	0,97
rbac	0,95	0,97	0,91	0,98	0,98	0,92	1	0,98	0,96
sanitize	0,92	0,95	0,98	1	0,97	0,94	0,98	0,97	0,96
socket-limit	0,94	0,93	0,97	1	0,96	0,99	0,96	0,96	0,96
stackleak	1,1	1,03	1,1	1,09	1,07	1,04	1,05	1,09	1,07
structleak	0,91	0,96	0,95	0,98	0,93	0,95	1	0,96	0,95
uderef	0,99	1,25	1,23	1,19	1,17	1,15	1,18	1,19	1,16

Tabulka 13.44: Relativní srovnání konfigurací vůči referenční konfiguraci original



Obrázek 13.6: Relativní srovnání konfigurací vůči referenční konfiguraci original

13.6.3.3 Zhodnocení testu

Z měření vyplynulo, že navazování spojení přes AF_UNIX sockety přítomnost Grsec neovlivňuje s výjimkou konfigurací autoconf-security (zhoršení o 24 %), uderef (zhoršení o 16 %) a stackleak (zhoršení o 7 %). První zmíněná je nadmnožinou zbylých dvou a její výsledek tomu i odpovídá. Velký rozptyl naměřených hodnot jednotlivých konfigurací pro jednu dvojici procesů vyšel i při tří násobném zvýšení počtu opakování testu a může za něj NMMP. **Systémům citlivým na zpoždění nelze doporučit použití voleb** `PAX_MEMORY_UDEREF`, `GRKERNSEC_CONFIG_PRIORITY_SECURITY` **a částečně i** `PAX_MEMORY_STACKLEAK`.

13.6.4 Test lmbench_bw_file_read_io_only

Program `bw_file_rd` testuje rychlost přístupu k externí paměti. Před testem je příkazem

```
dd if=/dev/urandom of=filename bs=1G
```

vytvořen soubor. Program `bw_file_rd` následně čte náhodně 300 MB z daného souboru po 64 kB blocích. Načtená data dále bere za posloupnost 4 B celých čísel, která **sčítá v číslo jedno**. Program využívá rozhraní poskytovaného systémem ke čtení souboru – systémové volání `read` – čili přesun souboru z Kernel Space do User Space obstarává jádro. NZPV představuje počet paralelně běžících procesů použitých pro sčítání. Nakonec řetězec `io_only` značí, že do měření jsou zahrnuty pouze I/O operace bez otevírání a zavírání souboru. Každý proces zapojený do měření provádí nezávisle čtení daného souboru. Rychlosti dosažené jednotlivými procesy se nakonec sčítají v rychlost jedinou.

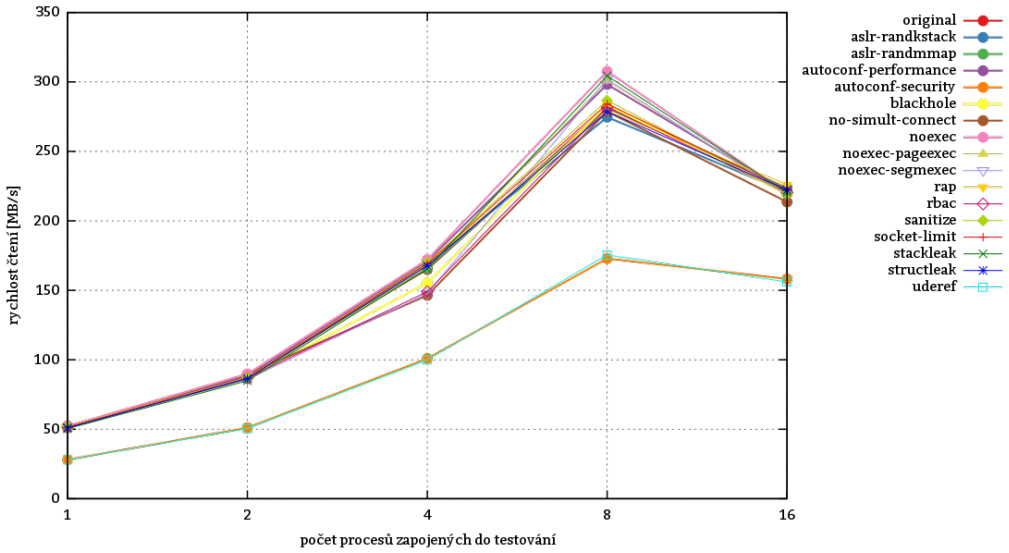
NZPV (x)	počet procesů zapojených do testování
jednotka NZPV	není (bezrozměrná veličina)
ZPV (y)	rychlost čtení
jednotka ZPV	MB/s
počet opakování měření	7
počítač spouštějící testovací příkaz	xenmv
testovací příkaz	`bw_file_rd -P ${X} -N 7 300M io_only testfile`

Tabulka 13.45: Charakteristiky testu lmbench_bw_file_read_io_only

13.6.4.1 Naměřené hodnoty pro jednotlivé konfigurace

měřená konfigurace	počet procesů zapojených do testování 1	2	4	8	16
original	52,05	88,52	165,1	282,1	223,4
aslr-randkstack	51,99	87,11	168,9	274,6	220,9
aslr-randmmap	50,90	88,00	165,8	281,2	223,4
autoconf-performance	52,05	88,49	170,9	298,2	221,0
autoconf-security	28,06	51,05	100,8	172,9	158,1
blackhole	51,38	86,58	155,3	283,3	219,3
no-simult-connect	51,62	89,57	146,3	278,7	213,5
noexec	52,25	89,81	172,0	307,5	219,9
noexec-pageexec	51,33	85,79	166,5	300,9	219,4
noexec-segmexec	51,35	86,01	147,5	301,9	218,4
rap	51,39	86,81	169,	279,2	226,0
rbac	51,57	86,85	149,0	281,4	220,2
sanitize	52,09	87,12	170,0	286,6	219,3
socket-limit	51,56	86,72	169,2	284,1	222,3
stackleak	50,53	84,96	165,5	304,3	220,4
structleak	50,96	86,45	167,6	278,5	222,0
uderef	28,04	50,55	99,9	175,4	155,9

Tabulka 13.46: Absolutní srovnání konfigurací

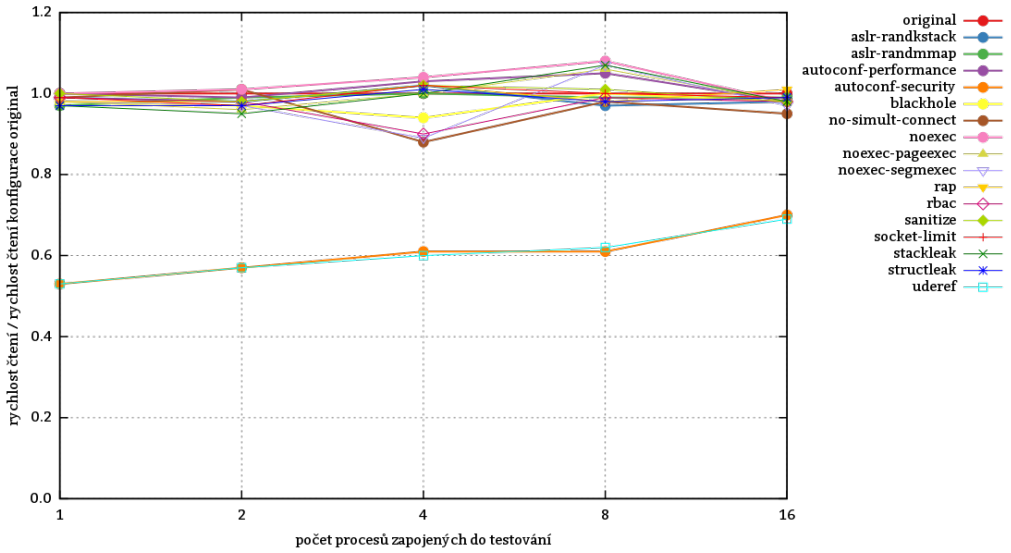


Obrázek 13.7: Absolutní srovnání konfigurací

13.6.4.2 Srovnání konfigurací

měřená konfigurace	počet procesů zapojených do testování 1	2	4	8	16	průměr
original	1	1	1	1	1	1
aslr-randkstack	0,99	0,98	1,02	0,97	0,98	0,98
aslr-randmmap	0,97	0,99	1	0,99	0,99	0,98
autoconf-performance	1	0,99	1,03	1,05	0,98	1,01
autoconf-security	0,53	0,57	0,61	0,61	0,7	0,6
blackhole	0,98	0,97	0,94	1	0,98	0,97
no-simult-connect	0,99	1,01	0,88	0,98	0,95	0,96
noexec	1	1,01	1,04	1,08	0,98	1,02
noexec-pageexec	0,98	0,96	1	1,06	0,98	0,99
noexec-segmexec	0,98	0,97	0,89	1,07	0,97	0,97
rap	0,98	0,98	1,02	0,98	1,01	0,99
rbac	0,99	0,98	0,9	0,99	0,98	0,96
sanitize	1	0,98	1,02	1,01	0,98	0,99
socket-limit	0,99	0,97	1,02	1	0,99	0,99
stackleak	0,97	0,95	1	1,07	0,98	0,99
structleak	0,97	0,97	1,01	0,98	0,99	0,98
uderef	0,53	0,57	0,6	0,62	0,69	0,6

Tabulka 13.47: Relativní srovnání konfigurací vůči referenční konfiguraci original



Obrázek 13.8: Relativní srovnání konfigurací vůči referenční konfiguraci original

13.6.4.3 Zhodnocení testu

Konfigurace autoconf-security a udereff, čili volba `PAX_MEMORY_UDEREF` rychlost čtení z externí paměti redukují o průměrně 40 %. Rychlost čtení kulminuje při NZPV rovné 8. To dává smysl, neb testovaný počítač – xenmv – disponuje právě 8 jádry (viz kapitola 11.2). 16 procesů již nedokáže přistupovat k externí paměti stejně efektivně jako 8. **Každopádně opět se potvrdil zásadní negativní dopad volby `PAX_MEMORY_UDEREF` na výkonnost systému.**

13.6.5 Test lmbench_bw_mem_rd

Program bw_mem testuje rychlost přístupu k hlavní paměti. V rámci tohoto měření alokuje blok o velikosti právě 1 MB, který následně opakovaně souvisle čte a měří přitom propustnost. Celkem je během testu přečteno 150 MB dat.

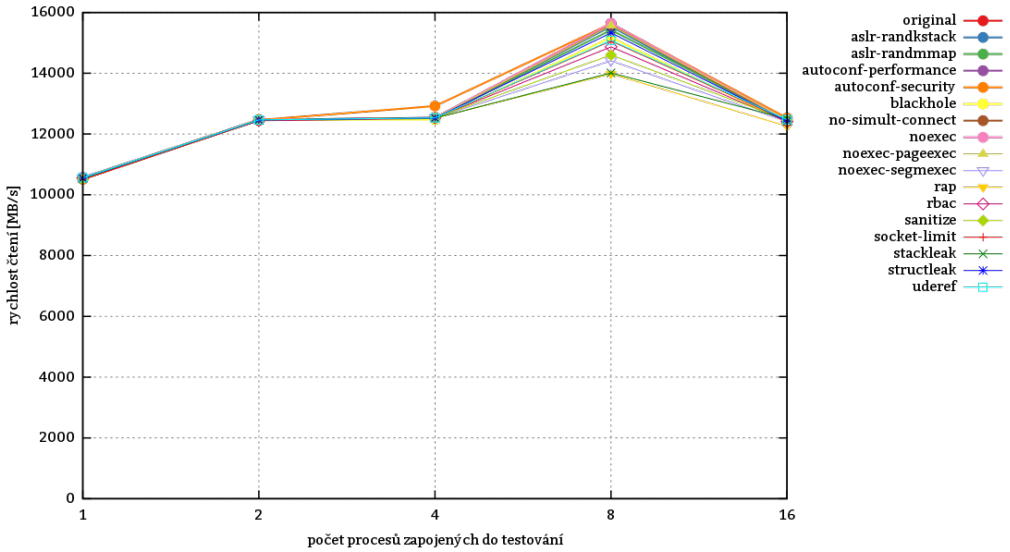
NZPV (x)	počet procesů zapojených do testování
jednotka NZPV	není (bezrozměrná veličina)
ZPV (y)	rychlost čtení
jednotka ZPV	MB/s
počet opakování měření	1
počítač spouštějící testovací příkaz	xenmv
testovací příkaz	`bw_mem -P ${X} 150M rd`

Tabulka 13.48: Charakteristiky testu lmbench_bw_mem_rd

13.6.5.1 Naměřené hodnoty pro jednotlivé konfigurace

měřená konfigurace	počet procesů zapojených do testování 1	2	4	8	16
original	10515,2	12460,18	12520,8	15568	12499,6
aslr-randkstack	10572,45	12466,85	12533,2	15540	12375,6
aslr-randmmap	10553,06	12458,64	12527,9	15420	12459,9
autoconf-performance	10539,16	12467,09	12530,8	15633	12453,1
autoconf-security	10515,91	12461,02	12919,7	15630	12527,9
blackhole	10544,58	12465,81	12476,2	15171	12388,5
no-simult-connect	10542,48	12450,18	12524,5	15588	12432,4
noexec	10555,43	12464,8	12531,9	15639	12462,7
noexec-pageexec	10571,74	12470,16	12530,9	15566	12476,2
noexec-segmexec	10551,18	12465,92	12527,4	14409	12408,9
rap	10549,54	12460,66	12526,2	13978	12256,2
rbac	10553,77	12441,52	12527,3	14867	12410,7
sanitize	10520,13	12464,4	12532,2	14598	12458,0
socket-limit	10502,90	12451,74	12529,6	15063	12441,9
stackleak	10532,81	12466,16	12525,7	14008	12498,4
structleak	10545,06	12461,83	12526,5	15328	12419,5
uderef	10557,15	12462,77	12528,8	15084	12468,6

Tabulka 13.49: Absolutní srovnání konfigurací

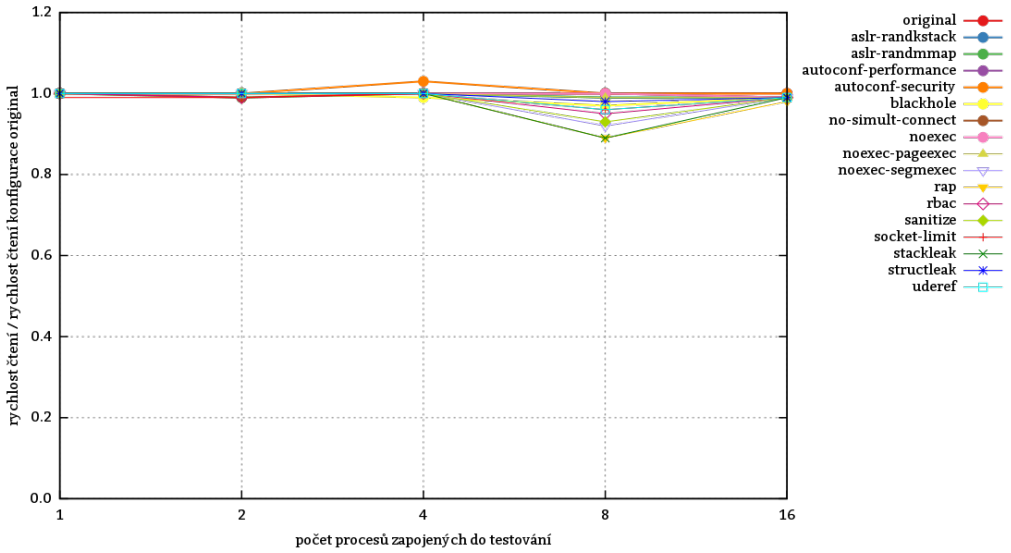


Obrázek 13.9: Absolutní srovnání konfigurací

13.6.5.2 Srovnání konfigurací

měřená konfigurace	počet procesů zapojených do testování 1	2	4	8	16	průměr
original	1	1	1	1	1	1
aslr-randkstack	1	1	1	0,99	0,99	0,99
aslr-randmmap	1	0,99	1	0,99	0,99	0,99
autoconf-performance	1	1	1	1	0,99	0,99
autoconf-security	1	1	1,03	1	1	1
blackhole	1	1	0,99	0,97	0,99	0,99
no-simult-connect	1	0,99	1	1	0,99	0,99
noexec	1	1	1	1	0,99	0,99
noexec-pageexec	1	1	1	0,99	0,99	0,99
noexec-segmexec	1	1	1	0,92	0,99	0,98
rap	1	1	1	0,89	0,98	0,97
rbac	1	0,99	1	0,95	0,99	0,98
sanitize	1	1	1	0,93	0,99	0,98
socket-limit	0,99	0,99	1	0,96	0,99	0,98
stackleak	1	1	1	0,89	0,99	0,97
structleak	1	1	1	0,98	0,99	0,99
uderef	1	1	1	0,96	0,99	0,99

Tabulka 13.50: Relativní srovnání konfigurací vůči referenční konfiguraci original



Obrázek 13.10: Relativní srovnání konfigurací vůči referenční konfiguraci original

13.6.5.3 Zhodnocení testu

Propustnost hlavní paměti při čtení je takřka nezávislá na zvolené konfiguraci vyjma situace, kdy je počet procesů zapojených do měření roven počtu systému dostupných procesorových jader (viz kapitola 11.2).

13.6.6 Test lmbench_bw_mem_wr

Program bw_mem testuje rychlost přístupu k hlavní paměti. V tomto měření alokuje 1 MB, do něhož opakovaně zapisuje a měří přitom propustnost. Během testu se zapíše 150 MB dat.

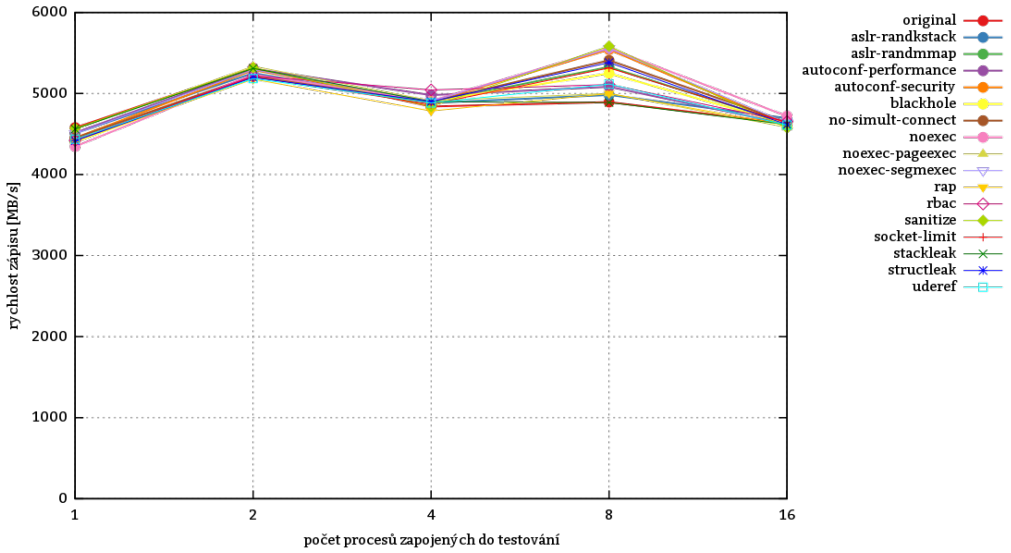
13.6.6.1 Naměřené hodnoty pro jednotlivé konfigurace

NZPV (x)	počet procesů zapojených do testování
jednotka NZPV	není (bezrozměrná veličina)
ZPV (y)	rychlost zápisu
jednotka ZPV	MB/s
počet opakování měření	1
počítač spouštějící testovací příkaz	xenmv
testovací příkaz	`bw_mem -P ${X} 150M wr`

Tabulka 13.51: Charakteristiky testu lmbench_bw_mem_wr

měřená konfigurace	**počet procesů zapojených do testování** 1	2	4	8	16
original	4578,9	5311,7	4839	4894	4630,9
aslr-randkstack	4509,3	5251,5	4883	4982	4693,8
aslr-randmmap	4412,5	5214,4	4895	5327	4631,4
autoconf-performance	4515,8	5305,7	4980	5083	4620,8
autoconf-security	4451,8	5287,7	4899	5540	4617,3
blackhole	4402,	5207,4	4904	5247	4625,0
no-simult-connect	4452,8	5217,8	4899	5409	4646,1
noexec	4346,7	5233,0	4909	5557	4720,3
noexec-pageexec	4558,3	5335,4	4912	4999	4582,1
noexec-segmexec	4504,7	5291,4	4905	5553	4638,6
rap	4416,6	5176,2	4783	4999	4608,5
rbac	4419,8	5216,5	5043	5110	4652,8
sanitize	4570,5	5317,8	4839	5583	4575,4
socket-limit	4435,6	5202,8	4866	5314	4640,5
stackleak	4561,2	5309,5	4899	4888	4617,2
structleak	4417,5	5197,1	4900	5381	4615,3
uderef	4435,9	5183,4	4884	5115	4609,9

Tabulka 13.52: Absolutní srovnání konfigurací

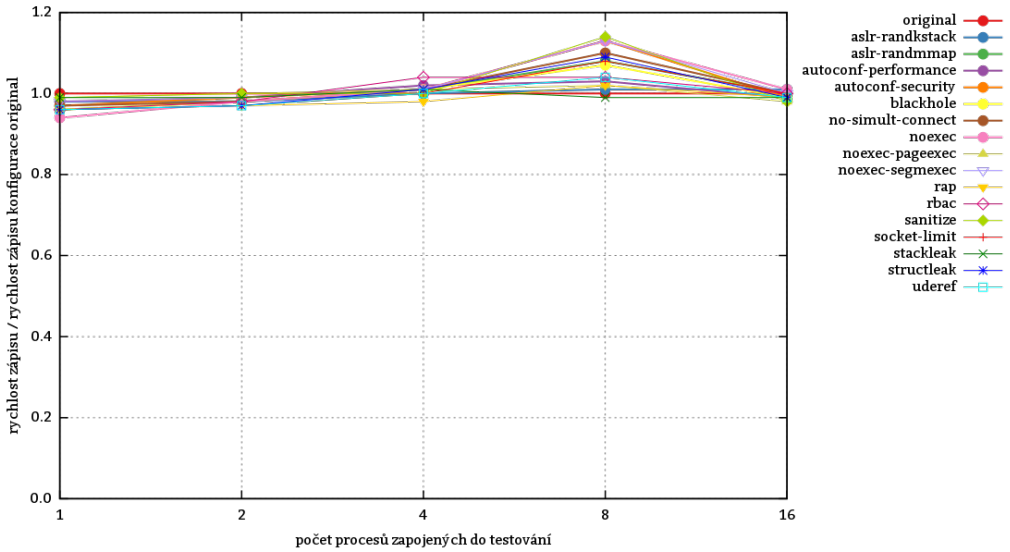


Obrázek 13.11: Absolutní srovnání konfigurací

13.6.6.2 Srovnání konfigurací

měřená konfigurace	počet procesů zapojených do testování 1	2	4	8	16	průměr
original	1	1	1	1	1	1
aslr-randkstack	0,98	0,98	1	1,01	1,01	0,99
aslr-randmmap	0,96	0,98	1,01	1,08	1	1
autoconf-performance	0,98	0,99	1,02	1,03	0,99	1
autoconf-security	0,97	0,99	1,01	1,13	0,99	1,01
blackhole	0,96	0,98	1,01	1,07	0,99	1
no-simult-connect	0,97	0,98	1,01	1,1	1	1,01
noexec	0,94	0,98	1,01	1,13	1,01	1,01
noexec-pageexec	0,99	1	1,01	1,02	0,98	1
noexec-segmexec	0,98	0,99	1,01	1,13	1	1,02
rap	0,96	0,97	0,98	1,02	0,99	0,98
rbac	0,96	0,98	1,04	1,04	1	1
sanitize	0,99	1	1	1,14	0,98	1,02
socket-limit	0,96	0,97	1	1,08	1	1
stackleak	0,99	0,99	1,01	0,99	0,99	0,99
structleak	0,96	0,97	1,01	1,09	0,99	1
uderef	0,96	0,97	1	1,04	0,99	0,99

Tabulka 13.53: Relativní srovnání konfigurací vůči referenční konfiguraci original



Obrázek 13.12: Relativní srovnání konfigurací vůči referenční konfiguraci original

13.6.6.3 Zhodnocení testu

Výsledek obdobný jako v testu 13.6.5. V podstatě se liší jen dosažené absolutní hodnoty. V 13.6.5 šlo o test rychlosti čtení hlavní paměti, zatímco tento test do ní zapisuje. Tomu odpovídá i pokles rychlostí: Tam, kde 1 vlákno dokázalo číst 10515 MB/s, se při zápisu přeneslo jen 4578 MB/s.

13.7 Testy nástrojem unixbench

Unixbench je, stejně tak jako LMBench, souborem jednoúčelových testů – mikrobenchmarků. Jednotlivé mikrobenchmarky provádějí nízkoúrovňové operace.

13.7.1 Test unixbench_pipe

Nechť *cyklus* sestává ze

1. zapsání 512 B do roury a
2. přečtení těchto 512 B z druhého jejího konce.

Test `unixbench_pipe` zjišťuje, kolik těchto cyklů stihne vykonat jeden proces za 10 s. Testu se účastní jediný proces, který zprávy zasílá sám sobě.

měřená veličina	počet cyklů, které 1 proces stihne provést za 10 s
jednotka měřené veličiny	není (bezrozměrná veličina)
počet opakování měření	7
počítač spouštějící testovací příkaz	xenmv
testovací příkaz	`pipe 10 2>&1 | tee /dev/stderr`

Tabulka 13.54: Charakteristiky testu unixbench_pipe

13.7.1.1 Naměřené hodnoty pro jednotlivé konfigurace

měřená konfigurace	počet cyklů, které 1 proces stihne provést za 10 s
original	18973647
aslr-randkstack	18339600
aslr-randmmap	19198212
autoconf-performance	18217627
autoconf-security	6347094
blackhole	20483032
no-simult-connect	19898579
noexec	19518807
noexec-pageexec	18807727
noexec-segmexec	18850635
rap	19571610
rbac	20516665
sanitize	20261066
socket-limit	20249876
stackleak	10162906
structleak	20417535
uderef	7682116

Tabulka 13.55: Absolutní srovnání konfigurací

13.7.1.2 Srovnání konfigurací

měřená konfigurace	počet cyklů, které 1 proces stihne provést za 10 s
original	1
aslr-randkstack	0,96
aslr-randmmap	1,01
autoconf-performance	0,96
autoconf-security	0,33
blackhole	1,07
no-simult-connect	1,04
noexec	1,02
noexec-pageexec	0,99
noexec-segmexec	0,99
rap	1,03
rbac	1,08
sanitize	1,06
socket-limit	1,06
stackleak	0,53
structleak	1,07
uderef	0,4

Tabulka 13.56: Relativní srovnání konfigurací vůči referenční konfiguraci original

13.7.1.3 Zhodnocení testu

Grsec výkonnost meziprocesové komunikace může významně ovlivnit, zejména pak v konfiguracích autoconf-security, stackleak a uderef.

Po vytvoření roury

```
pipe(pvec);
```

se neustále opakují funkce standardní knihovny jazyka C

```
write(pvec[1], buf, sizeof(buf)) != sizeof(buf)
```

a

```
read(pvec[0], buf, sizeof(buf)) != sizeof(buf)
```

transformující se na systémová volání. To zesiluje vliv volby `PAX_MEMORY_STACKLEAK`, která nutí jádro po každém systémovém volání čistit kernelspace-stack. Dále se významně negativně projevily konfigurace uderef a též autoconf-security obsahující jak `PAX_MEMORY_UDEREF`, tak i `PAX_MEMORY_STACKLEAK`. Pokles výkonnosti způsobený těmito volbami je až 67 %.

13.7.2 Test unixbench_execl

Test `unixbench_execl` měří počet volání `execl()` uskutečnitelných na daném systému za 10 s. Funkce `execl()` provádí obdobnou činnost jako `execve()` – nahrazuje proces procesem novým, což dokládá závěrečná část funkce `main()` tohoto testu zobrazená v C kódu 13.1. Výstupem tohoto testu je počet úspěšně provedených nahrazení procesu – úspěšných volání `execl()` – za 10 s.

```
/* ... */
    sprintf(count_str, "%lu", ++iter); /* increment the execl counter */
    sprintf(start_str, "%lu", (unsigned long) start_time);
    time(&this_time);
    if (this_time - start_time >= duration) { /* time has run out */
        fprintf(stderr, "COUNT|%lu|1|lps\n", iter);
        exit(0);
    }
    execl(fullpath, fullpath, "0", dur_str, count_str, start_str, (void *) 0);
    fprintf(stderr, "Exec failed at iteration %lu\n", iter);
    perror("Reason");
    exit(1);
}
```

C kód 13.1: Nahrazení stávajícího procesu voláním funkce `execl()` v rámci testu `unixbench_execl`

měřená veličina	počet úspěšně provedených nahrazení procesu
jednotka měřené veličiny	není (bezrozměrná veličina)
počet opakování měření	7
počítač spouštějící testovací příkaz	xenmv
testovací příkaz	`execl 10 2>&1 | tee /dev/stderr`

Tabulka 13.57: Charakteristiky testu unixbench_execl

13.7.2.1 Naměřené hodnoty pro jednotlivé konfigurace

měřená konfigurace	počet úspěšně provedených nahrazení procesu
original	39493
aslr-randkstack	40469
aslr-randmmap	40082
autoconf-performance	35075
autoconf-security	29426
blackhole	41134
no-simult-connect	39752
noexec	38262
noexec-pageexec	36755
noexec-segmexec	37759
rap	39527
rbac	39939
sanitize	40486
socket-limit	40465
stackleak	38626
structleak	39829
uderef	33066

Tabulka 13.58: Absolutní srovnání konfigurací

13.7.2.2 Srovnání konfigurací

měřená konfigurace	počet úspěšně provedených nahrazení procesu
original	1
aslr-randkstack	1,02
aslr-randmmap	1,01
autoconf-performance	0,88
autoconf-security	0,74
blackhole	1,04
no-simult-connect	1
noexec	0,96
noexec-pageexec	0,93
noexec-segmexec	0,95
rap	1
rbac	1,01
sanitize	1,02
socket-limit	1,02
stackleak	0,97
structleak	1
uderef	0,83

Tabulka 13.59: Relativní srovnání konfigurací vůči referenční konfiguraci original

13.7.2.3 Zhodnocení testu

Počet volání `execl()` uskutečnitelných na daném systému za 10 s Grsecurity snižuje. Výrazněji pak v konfiguracích uderef a autoconf-security.

13.7.3 Test unixbench_context1

Test `unixbench_context1` vytvoří dva procesy. Následně měří, kolikrát se tyto procesy vystřídají v inkrementaci celého čísla, které si vzájemně posílají prostřednictvím roury. Po přečtení hodnoty ji proces vždy inkrementuje a ihned odesílá skrze rouru zpět. Test probíhá 10 s. Na počátku má jimi předávané číslo hodnotu 0, na konci představuje výsledek testu.

13.7.3.1 Naměřené hodnoty pro jednotlivé konfigurace

měřená veličina	počet výměn hodnoty skrze rouru za 10 s
jednotka měřené veličiny	není (bezrozměrná veličina)
počet opakování měření	7
počítač spouštějící testovací příkaz	xenmv
testovací příkaz	`context1 10 2>&1 | tee /dev/stderr`

Tabulka 13.60: Charakteristiky testu unixbench_context1

měřená konfigurace	**počet výměn hodnoty skrze rouru za 10 s**
original	1332799
aslr-randkstack	1378279
aslr-randmmap	1376239
autoconf-performance	1196598
autoconf-security	998291
blackhole	1386173
no-simult-connect	1359749
noexec	1193432
noexec-pageexec	1160789
noexec-segmexec	1161787
rap	1379779
rbac	1387079
sanitize	1372651
socket-limit	1371357
stackleak	1223656
structleak	1323867
uderef	1086590

Tabulka 13.61: Absolutní srovnání konfigurací

13.7.3.2 Srovnání konfigurací

měřená konfigurace	počet výměn hodnoty skrze rouru za 10 s
original	1
aslr-randkstack	1,03
aslr-randmmap	1,03
autoconf-performance	0,89
autoconf-security	0,74
blackhole	1,04
no-simult-connect	1,02
noexec	0,89
noexec-pageexec	0,87
noexec-segmexec	0,87
rap	1,03
rbac	1,04
sanitize	1,02
socket-limit	1,02
stackleak	0,91
structleak	0,99
uderef	0,81

Tabulka 13.62: Relativní srovnání konfigurací vůči referenční konfiguraci original

13.7.3.3 Zhodnocení testu

Střídání dvojice procesů v inkrementaci celého čísla patch Grsecurity nezrychluje. Největší negativní výkonnostní dopad má, jako i ve většině předešlých testech, volba `PAX_MEMORY_UDEREF`. Poprvé se však negativně projevily dosud nevýrazné konfigurace noexec, noexec-pageexec a noexec-segmexec. Tyto by měly přinášet podporu NX bitu čili nespustitelných stránek respektive v případě systému s podporou x86-64 instrukcí ji vynutit v praxi používat.

13.7.4 Test unixbench_spawn

Test `unixbench_spawn` měří počet procesů, které 1 proces v daném časovém limitu (10 s) stihne vytvořit voláním `fork()`. Potomci procesu jsou po svém úspěšném vytvoření ihned ukončeni.

měřená veličina	počet procesů vytvořených voláním fork za 10 s
jednotka měřené veličiny	není (bezrozměrná veličina)
počet opakování měření	7
počítač spouštějící testovací příkaz	xenmv
testovací příkaz	`spawn 10 2>&1 | tee /dev/stderr`

Tabulka 13.63: Charakteristiky testu unixbench_spawn

13.7.4.1 Naměřené hodnoty pro jednotlivé konfigurace

měřená konfigurace	počet procesů vytvořených voláním fork za 10 s
original	105935
aslr-randkstack	106829
aslr-randmmap	104403
autoconf-performance	90246
autoconf-security	82409
blackhole	106906
no-simult-connect	108566
noexec	93760
noexec-pageexec	92895
noexec-segmexec	92341
rap	106407
rbac	105463
sanitize	105211
socket-limit	108866
stackleak	100176
structleak	109220
uderef	94535

Tabulka 13.64: Absolutní srovnání konfigurací

13.7.4.2 Srovnání konfigurací

měřená konfigurace	počet procesů vytvořených voláním fork za 10 s
original	1
aslr-randkstack	1
aslr-randmmap	0,98
autoconf-performance	0,85
autoconf-security	0,77
blackhole	1
no-simult-connect	1,02
noexec	0,88
noexec-pageexec	0,87
noexec-segmexec	0,87
rap	1
rbac	0,99
sanitize	0,99
socket-limit	1,02
stackleak	0,94
structleak	1,03
uderef	0,89

Tabulka 13.65: Relativní srovnání konfigurací vůči referenční konfiguraci original

13.7.4.3 Zhodnocení testu

Výsledek je podobný výsledku dosaženému měřením v kapitole 13.7.3. V tomto případě není dopad volby `PAX_MEMORY_UDEREF` na výkonnost tak patrný jako ve zmíněném měření. Patrně to bude způsobeno faktem, že v tomto případě přece jen proběhne o poznání méně systémových volání a tedy i výměn dat mezi User Space a Kernel Space, které právě volba `PAX_MEMORY_UDEREF` brzdí. Konfigurace nějakým způsobem spjaté s NX bitem způsobují zhoršení výkonnosti o cca 13 %.

13.7.5 Test unixbench_syscall

V rámci testu `unixbench_syscall` jednoduchý program opakovaně volá

- **funkci `close()`**,

```
while (1) { close(dup(0)); iter++; }
```

- **funkci `getpid()`**,

```
while (1) { getpid(); iter++; }
```

- **funkci `exec()`** nebo

```
while (1) {
    pid_t pid = fork();
    if (pid < 0) {
        fprintf(stderr,"%s: fork failed\n", argv[0]);
        exit(1);
    } else if (pid == 0) {
        execl("/bin/true", "/bin/true", (char *) 0);
        fprintf(stderr,"%s: exec /bin/true failed\n", argv[0]);
        exit(1);
    } else {
        if (waitpid(pid, NULL, 0) < 0) {
            fprintf(stderr,"%s: waitpid failed\n", argv[0]);
            exit(1);
        }
    }
    iter++;
}
```

- **všechny výše uvedené funkce při přibližně stejné četnosti volání každé z nich**.

```
while (1) { close(dup(0)); getpid(); getuid(); umask(022); iter++; }
```

Na jejich volání má omezený čas. Výstupem testu je **počet dokončených volání za 10 s**. Pochopitelně zde platí přímá úměra: Čím je počet dokončených volání v daném čase menší, tím větší jsou náklady na přepnutí z user mode do kernel mode a zase zpět.

NZPV (x)	typ systémových volání
jednotka NZPV	není (bezrozměrná veličina)
ZPV (y)	počet dokončených volání daného typu za 10 s
jednotka ZPV	není (bezrozměrná veličina)
počet opakování měření	7
počítač spouštějící testovací příkaz	xenmv
testovací příkaz	`syscall 10 ${X} 2>&1 | tee /dev/stderr`

Tabulka 13.66: Charakteristiky testu unixbench_syscall

13.7.5.1 Naměřené hodnoty pro jednotlivé konfigurace

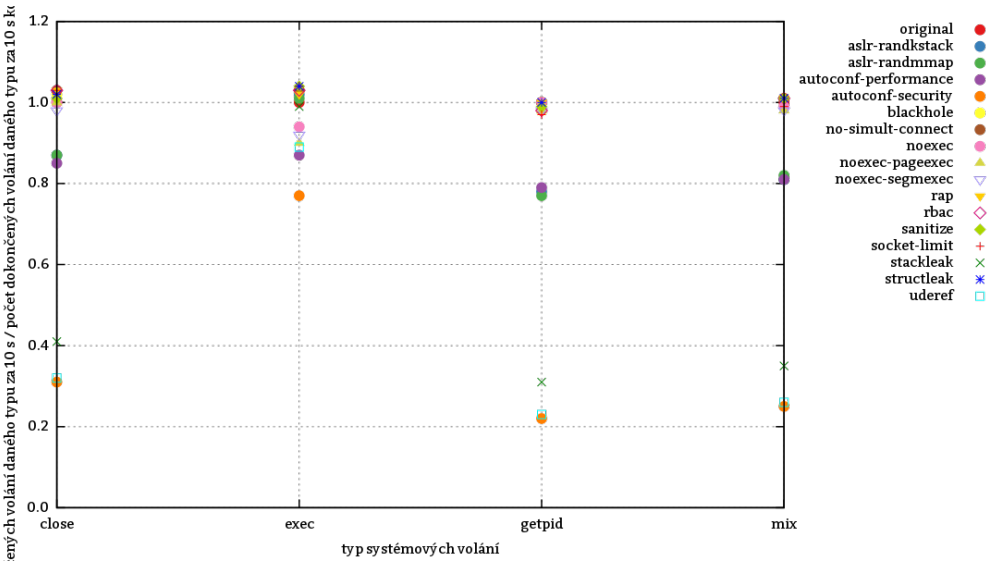
měřená konfigurace	typ systémových volání close	exec	getpid	mix
original	52973925	31035	169258450	28084062
aslr-randkstack	46224722	31933	132360318	22926830
aslr-randmmap	46375352	31465	131079846	23081849
autoconf-performance	45438179	27097	133987421	22948095
autoconf-security	16616653	24039	37957026	7196484
blackhole	54519547	32271	167216810	28489886
no-simult-connect	55059974	32161	169811437	28486342
noexec	53438172	29196	170822437	28051658
noexec-pageexec	53455420	28011	166794907	27725934
noexec-segmexec	52186998	28741	166332191	27798614
rap	54881484	31835	169919531	28394941
rbac	54875862	32125	166310413	28482315
sanitize	53848868	32292	167835331	28519992
socket-limit	54789769	32569	164659031	28044317
stackleak	21917115	30806	53731186	9949681
structleak	54342763	32301	169996077	28461844
uderef	17405268	27896	39851823	7565131

Tabulka 13.67: Absolutní srovnání konfigurací

13.7.5.2 Srovnání konfigurací

měřená konfigurace	typ systémových volání close	exec	getpid	mix	průměr
original	1	1	1	1	1
aslr-randkstack	0,87	1,02	0,78	0,81	0,87
aslr-randmmap	0,87	1,01	0,77	0,82	0,86
autoconf-performance	0,85	0,87	0,79	0,81	0,83
autoconf-security	0,31	0,77	0,22	0,25	0,38
blackhole	1,02	1,03	0,98	1,01	1,01
no-simult-connect	1,03	1,03	1	1,01	1,01
noexec	1	0,94	1	0,99	0,98
noexec-pageexec	1	0,9	0,98	0,98	0,96
noexec-segmexec	0,98	0,92	0,98	0,98	0,96
rap	1,03	1,02	1	1,01	1,01
rbac	1,03	1,03	0,98	1,01	1,01
sanitize	1,01	1,04	0,99	1,01	1,01
socket-limit	1,03	1,04	0,97	0,99	1
stackleak	0,41	0,99	0,31	0,35	0,51
structleak	1,02	1,04	1	1,01	1,01
uderef	0,32	0,89	0,23	0,26	0,42

Tabulka 13.68: Relativní srovnání konfigurací vůči referenční konfiguraci original



Obrázek 13.13: Relativní srovnání konfigurací vůči referenční konfiguraci original

13.7.5.3 Zhodnocení testu

Počet dokončených volání za 10 s Grsec zpravidla snižuje. Konfigurace stackleak, uderef a autoconf-security mají naprosto zásadní dopad na rychlost obsluhy systémových volání vyjma volání `exec()` a dokáží výkonnost systému ponížit až na 38 % jeho původní výkonnosti. Poprvé se v měření výrazněji projevila konfigurace aslr-randmmap, která má za úkol znáhodňovat umístění jakýchkoliv mapování programů včetně mapování anonymních. Více viz kapitola 9.1.5.

13.7.6 Test unixbench_fstime

Test unixbench_fstime zjišťuje datový tok při kopírování za tímto účelem vytvořeného dočasného souboru do jiného umístění na disku při různých velikostech bloku. Nezávisle proměnnou v tomto testu je právě velikost bloku. Kopírování trvá 10 s.

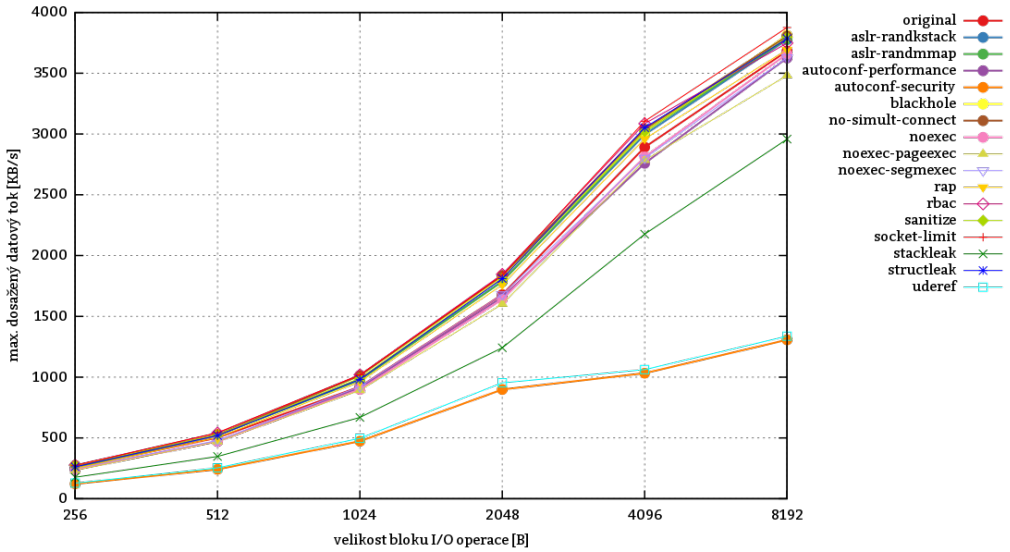
NZPV (x)	velikost bloku I/O operace
jednotka NZPV	B
ZPV (y)	max. dosažený datový tok
jednotka ZPV	KB/s
počet opakování měření	3
počítač spouštějící testovací příkaz	xenmv
testovací příkaz	`fstime -c -b ${X} -t 10 2>&1 | tee /dev/stderr`

Tabulka 13.69: Charakteristiky testu unixbench_fstime

13.7.6.1 Naměřené hodnoty pro jednotlivé konfigurace

měřená konfigurace	velikost bloku I/O operace [B] 256	512	1024	2048	4096	8192
original	257,3	502,1	921	1678	2891	3682
aslr-randkstack	263,5	520,7	984	1789	2994	3775
aslr-randmmap	259,9	522,1	976	1786	3015	3781
autoconf-performance	239,8	471,8	913	1656	2758	3625
autoconf-security	122,6	242,1	473	898	1032	1306
blackhole	273,1	524,0	1010	1824	3019	3811
no-simult-connect	271,2	526,2	1013	1835	3039	3804
noexec	240,4	471,1	898	1639	2811	3659
noexec-pageexec	233,3	469,4	891	1600	2785	3480
noexec-segmexec	247,8	479,1	923	1679	2801	3627
rap	249,8	503,5	961	1759	2959	3694
rbac	277,2	540,7	1019	1846	3085	3747
sanitize	272,0	528,0	990	1811	3022	3793
socket-limit	274,0	543,0	1018	1838	3105	3875
stackleak	176,3	347,4	668	1242	2176	2958
structleak	262,3	519,5	981	1809	3052	3784
uderef	128,9	253,8	497	953	1062	1337

Tabulka 13.70: Absolutní srovnání konfigurací

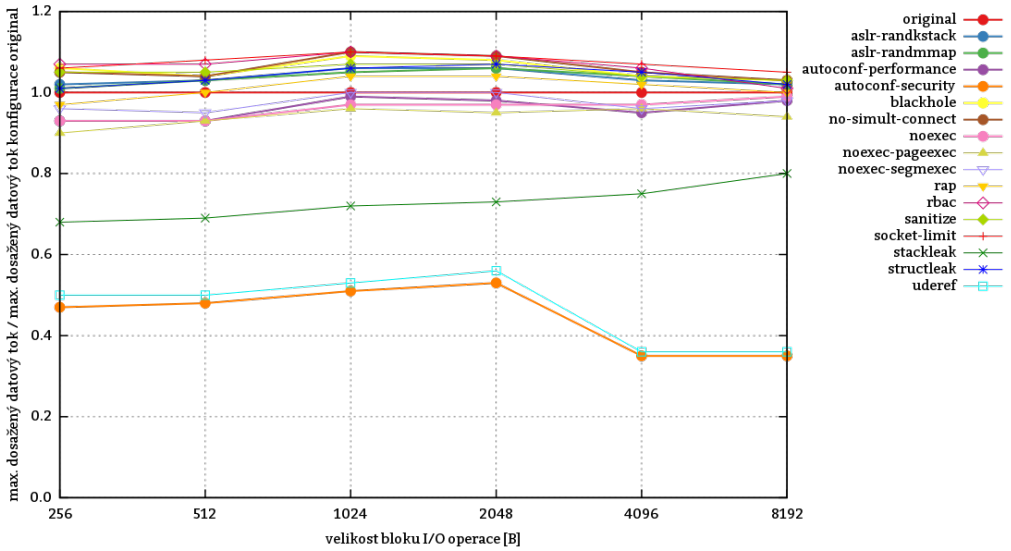


Obrázek 13.14: Absolutní srovnání konfigurací

13.7.6.2 Srovnání konfigurací

měřená konfigurace	velikost bloku I/O operace [B] 256	512	1024	2048	4096	8192	průměr
original	1	1	1	1	1	1	1
aslr-randkstack	1,02	1,03	1,06	1,06	1,03	1,02	1,03
aslr-randmmap	1,01	1,03	1,05	1,06	1,04	1,02	1,03
autoconf-performance	0,93	0,93	0,99	0,98	0,95	0,98	0,96
autoconf-security	0,47	0,48	0,51	0,53	0,35	0,35	0,44
blackhole	1,06	1,04	1,09	1,08	1,04	1,03	1,05
no-simult-connect	1,05	1,04	1,1	1,09	1,05	1,03	1,06
noexec	0,93	0,93	0,97	0,97	0,97	0,99	0,96
noexec-pageexec	0,9	0,93	0,96	0,95	0,96	0,94	0,94
noexec-segmexec	0,96	0,95	1	1	0,96	0,98	0,97
rap	0,97	1	1,04	1,04	1,02	1	1,01
rbac	1,07	1,07	1,1	1,09	1,06	1,01	1,06
sanitize	1,05	1,05	1,07	1,07	1,04	1,03	1,05
socket-limit	1,06	1,08	1,1	1,09	1,07	1,05	1,07
stackleak	0,68	0,69	0,72	0,73	0,75	0,8	0,72
structleak	1,01	1,03	1,06	1,07	1,05	1,02	1,04
uderef	0,5	0,5	0,53	0,56	0,36	0,36	0,46

Tabulka 13.71: Relativní srovnání konfigurací vůči referenční konfiguraci original



Obrázek 13.15: Relativní srovnání konfigurací vůči referenční konfiguraci original

13.7.6.3 Zhodnocení testu

Přítomnost Grsecurity v některých konfiguracích přinesla dokonce nárůst výkonnosti. Vyjma konfigurací uderef a stackleak však lze odchylky od výsledku referenční konfigurace přičíst NMMP. Významnější dopad je pozorovatelný u konfigurací stackleak, uderef a autoconf-security.

Konfigurace uderef obsahuje aktivovanou volbu `PAX_MEMORY_UDEREF`. Tato volba má naprosto zásadní vliv na výkonnost I/O operací, kdy aktivace jí samotné srazí výkon na 46 %. Konfigurace stackleak vynucením mazání kernelspace-stack před každým předáním obsluhy uživatelskému programu způsobuje taktéž zpomalení, avšak průměrně jen o 28 %.

13.8 Testy nástrojem sysbench

Nástroj `sysbench` poskytuje funkce pro testování celé škály systémových komponent. Příkazy vyskytující se v této kapitole přejímají mnohé parametry. Z důvodu úspory prostoru se popis jejich významu vyskytuje v tomto dokumentu pouze jednou a to v tabulce 13.72.

Přepínač	Význam
`--test`	určení testu z množiny všech testů poskytovaných nástrojem `sysbench`
`--file-num`	počet souborů, nad kterými bude prováděn test
`--file-total-size`	celková velikost souborů, nad kterými bude prováděn test
`--file-block-size`	velikost bloku, kterým budou prováděny jednotlivé operace
`--max-time`	omezení trvání testu shora (v sekundách)
`--file-extra-flags`	Volba „direct“ minimalizuje vliv mezipaměti při přístupu k souboru, který se používá.
`--file-test-mode`	charakteristika přístupu k souboru (náhodné/sekvenční čtení/zápis)
`--max-requests`	omezení počtu operací

Tabulka 13.72: Parametry příkazu `sysbench`

13.8.1 Test sysbench_cpu

Test `sysbench_cpu` nechá jím prověřovaný počítač hledat největší prvočíslo menší než 100000 . Algoritmus hledání nejvyššího prvočísla, který program používá je následující:

1. Program vezme číslo n a to se pokouší postupně dělit číslem m, přičemž $m \in (2, \sqrt{n})$[3].
2. Vyjde-li podíl $\frac{n}{m}$ pro alespoň jedno m beze zbytku, potom program našel dělitele čísla n a číslo n proto není prvočíslo. V ten okamžik program končí se zkoumáním čísla n.
3. Nevyjde-li podíl pro žádné m beze zbytku, potom číslo n není prvočíslo. Po vyzkoušení všech m program končí se zkoumáním čísla n.
4. Nebylo-li n posledním zkoumaným číslem, potom program přechází k číslu $n + 1$ a pokračuje opět bodem 1.

NZPV (x)	počet úlohu zpracovávajících vláken
jednotka NZPV	není (bezrozměrná veličina)
ZPV (y)	doba potřebná k nalezení max. prvočísla < 100000
jednotka ZPV	s
počet opakování měření	3
počítač spouštějící testovací příkaz	xenmv
testovací příkaz	`sysbench --test=cpu --cpu-max-prime=100000 --num-threads=${X} run`

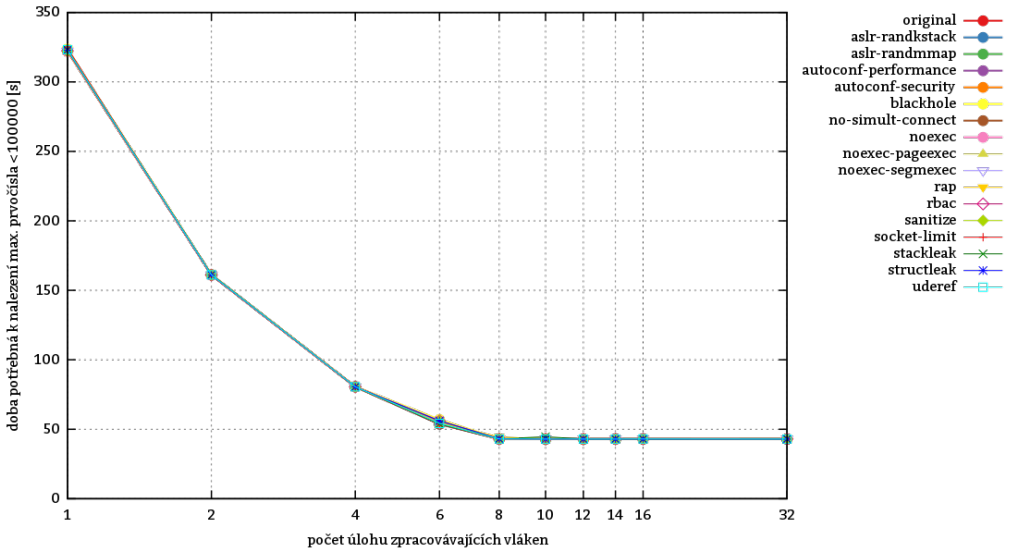
Tabulka 13.73: Charakteristiky testu sysbench_cpu

13.8.1.1 Naměřené hodnoty pro jednotlivé konfigurace

[3] Program postupně vybírá m od nejmenšího možného po největší možné.

měřená konfigurace	počet úlohu zpracovávajících vláken									
	1	2	4	6	8	10	12	14	16	32
original	322,348	161,011	80,511	53,686	43,01	42,971	42,956	42,955	42,962	42,97
aslr-randkstack	322,817	161,282	80,54	53,69	43,07	42,968	42,96	42,959	42,952	42,95
aslr-randmmap	322,185	161,278	80,607	54,586	43,087	42,987	42,954	42,96	42,953	42,98
autoconf-performance	322,722	161,073	80,67	56,019	42,991	42,953	42,958	42,958	42,952	42,953
autoconf-security	322,025	161,021	80,518	56,029	42,97	42,973	42,963	42,959	42,959	42,971
blackhole	323,653	161,197	80,691	56,994	43,131	42,953	42,953	42,954	42,957	43,004
no-simult-connect	322,67	161,007	80,509	53,784	42,986	42,999	43,03	43,022	42,989	42,961
noexec	321,99	161,009	80,502	56,026	42,991	42,973	42,961	42,957	42,955	42,955
noexec-pageexec	322,366	160,999	80,505	54,58	43,024	42,966	42,957	42,958	42,955	42,96
noexec-segmexec	322,143	161,079	80,509	54,472	43,008	42,977	42,957	42,955	42,958	42,956
rap	323,341	161,055	80,506	54,571	44,514	42,953	42,952	43,032	43,014	42,966
rbac	322,353	160,988	80,529	56,019	43,017	42,95	42,953	42,989	43,014	43,018
sanitize	322,12	161,01	80,506	56,05	43,007	42,962	42,957	43,041	43,041	43,008
socket-limit	324,507	161,002	80,509	53,723	43,116	42,985	42,956	42,954	42,958	42,95
stackleak	323,147	161,373	80,503	53,718	43,095	44,606	42,955	42,98	42,96	43,012
structleak	322,842	160,998	80,504	56,029	43,131	42,961	42,952	42,953	42,972	42,97
uderef	323,224	161,13	80,531	54,183	43,03	42,968	42,957	42,961	42,995	42,96

Tabulka 13.74: Absolutní srovnání konfigurací

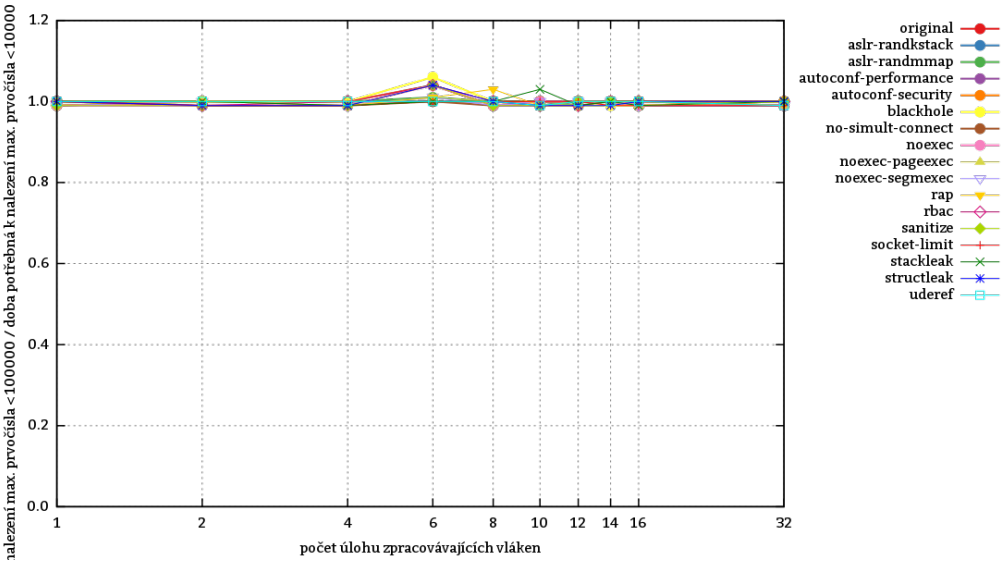


Obrázek 13.16: Absolutní srovnání konfigurací

13.8.1.2 Srovnání konfigurací

měřená konfigurace	počet úlohu zpracovávajících vláken										průměr
	1	2	4	6	8	10	12	14	16	32	
original	1	1	1	1	1	1	1	1	1	1	1
aslr-randkstack	1	1	1	1	1	0,99	1	1	0,99	0,99	0,99
aslr-randmmap	0,99	1	1	1,01	1	1	0,99	1	0,99	1	0,99
autoconf-performance	1	1	1	1,04	0,99	0,99	1	1	0,99	0,99	1
autoconf-security	0,99	1	1	1,04	0,99	1	1	1	0,99	1	1
blackhole	1	1	1	1,06	1	0,99	0,99	0,99	0,99	1	1
no-simult-connect	1	0,99	0,99	1	0,99	1	1	1	1	0,99	0,99
noexec	0,99	0,99	0,99	1,04	0,99	1	1	1	0,99	0,99	0,99
noexec-pageexec	1	0,99	0,99	1,01	1	0,99	1	1	0,99	0,99	0,99
noexec-segmexec	0,99	1	0,99	1,01	0,99	1	1	1	0,99	0,99	0,99
rap	1	1	0,99	1,01	1,03	0,99	0,99	1	1	0,99	1
rbac	1	0,99	1	1,04	1	0,99	0,99	1	1	1	1
sanitize	0,99	0,99	0,99	1,04	0,99	0,99	1	1	1	1	0,99
socket-limit	1	0,99	0,99	1	1	1	1	0,99	0,99	0,99	0,99
stackleak	1	1	0,99	1	1	1,03	0,99	1	0,99	1	1
structleak	1	0,99	0,99	1,04	1	0,99	0,99	0,99	1	1	0,99
uderef	1	1	1	1	1	0,99	1	1	1	0,99	0,99

Tabulka 13.75: Relativní srovnání konfigurací vůči referenční konfiguraci original



Obrázek 13.17: Relativní srovnání konfigurací vůči referenční konfiguraci original

13.8.1.3 Zhodnocení testu

Test nezjistil mezi konfiguracemi rozdíl větší, nežli by mohla být chyba měření. Výpočet škáluje s počtem procesorových jader. Přidávání vláken po dosažení počtu systému dostupných procesorových jader již nepřináší žádný užitek.

13.8.2 Test sysbench_fileio_rndrd_s

Test `sysbench_fileio_rndrd_s` měří datovou propustnost při přístupu k souborům. Je prováděno 100000 přístupů/operací nad náhodně volenými soubory. Jedná se o 100 za tímto účelem vytvořených souborů. Každý jeden soubor je velký 2 kB. Po ukončení testu jsou z FS odstraněny. NZPV je velikost bloku – 512, 1024, 1536[4] a 2048. Vzhledem k tomu, že mezi hodnotami NZPV je tak zařazena jedna, která pravděpodobně způsobí pokles výkonu, bylo přijato rozhodnutí, že hodnoty v grafech reprezentujících naměřená data budou spojeny pouze přerušovanými čarami, což by mělo čtenáři připomenout, že spojnice mezi naměřenými hodnotami „nemají význam“ a byly zachovány pouze v zájmu

jeho lepší orientace.

NZPV (x)	velikost bloku
jednotka NZPV	B
ZPV (y)	propustnost
jednotka ZPV	Mb/s
počet opakování měření	3
počítač spouštějící testovací příkaz	xenmv

testovací příkaz	`sysbench --test=fileio --file-num=100 --file-total-size=200K`
	`↪ --max-requests=100000 --max-time=200`
	`↪ --file-extra-flags=direct --file-block-size=${X}`
	`↪ --file-test-mode=rndrd run`

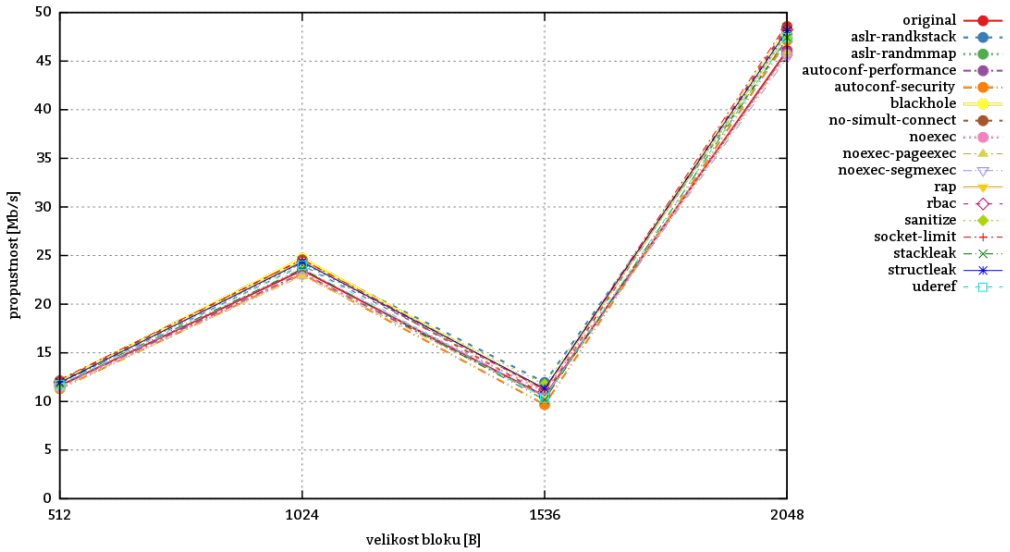
Tabulka 13.76: Charakteristiky testu sysbench_fileio_rndrd_s

13.8.2.1 Naměřené hodnoty pro jednotlivé konfigurace

měřená konfigurace	velikost bloku [B] 512	1024	1536	2048
original	11,626	23,492	10,56	46,091
aslr-randkstack	12,091	24,539	11,95	47,206
aslr-randmmap	12,05	24,271	11,25	47,857
autoconf-performance	11,61	23,18	10,49	45,928
autoconf-security	11,305	23,095	9,67	47,131
blackhole	12,111	24,737	11,26	48,336
no-simult-connect	12,121	24,566	10,68	48,552
noexec	11,422	23,071	11,05	45,696
noexec-pageexec	11,399	22,916	10,49	45,771
noexec-segmexec	11,579	23,66	10,57	45,347
rap	11,93	24,481	11,18	47,732
rbac	11,948	23,999	11,23	48,236
sanitize	11,859	24,188	11,75	47,342
socket-limit	12,24	24,594	11,20	48,851
stackleak	11,875	23,63	10,17	47,423
structleak	11,953	24,343	11,33	48,114
uderef	11,753	23,987	10,35	47,413

Tabulka 13.77: Absolutní srovnání konfigurací

[4] Velikost bloku 1536 byla doplněna až na dodatečně. Autor, jsa vědom toho, že velikost bloku, která není mocninou dvou, se v praxi nepoužívá, neboť práce s ní bude z principu značně neefektivní, dostal explicitní požehnání k jeho zařazení do testu od vedoucího této práce. Oba se shodli na tom, že naměřený výsledek bude přínosný minimálně v tom, že ukáže, jak moc poklesne výkonnost systému.

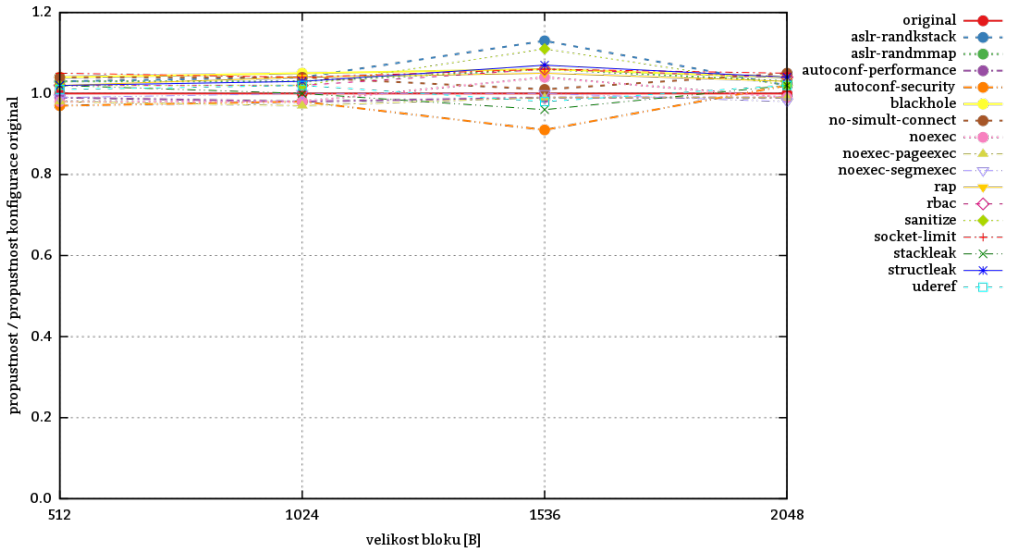


Obrázek 13.18: Absolutní srovnání konfigurací

13.8.2.2 Srovnání konfigurací

měřená konfigurace	velikost bloku [B] 512	1024	1536	2048	průměr
original	1	1	1	1	1
aslr-randkstack	1,03	1,04	1,13	1,02	1,05
aslr-randmmap	1,03	1,03	1,06	1,03	1,03
autoconf-performance	0,99	0,98	0,99	0,99	0,98
autoconf-security	0,97	0,98	0,91	1,02	0,97
blackhole	1,04	1,05	1,06	1,04	1,04
no-simult-connect	1,04	1,04	1,01	1,05	1,03
noexec	0,98	0,98	1,04	0,99	0,99
noexec-pageexec	0,98	0,97	0,99	0,99	0,98
noexec-segmexec	0,99	1	1	0,98	0,99
rap	1,02	1,04	1,05	1,03	1,03
rbac	1,02	1,02	1,06	1,04	1,03
sanitize	1,02	1,02	1,11	1,02	1,04
socket-limit	1,05	1,04	1,06	1,05	1,05
stackleak	1,02	1	0,96	1,02	1
structleak	1,02	1,03	1,07	1,04	1,04
uderef	1,01	1,02	0,98	1,02	1

Tabulka 13.78: Relativní srovnání konfigurací vůči referenční konfiguraci original



Obrázek 13.19: Relativní srovnání konfigurací vůči referenční konfiguraci original

13.8.2.3 Zhodnocení testu

Test ukázal, že propustnost při náhodném čtení velkého množství malých souborů limituje něco zcela jiného nežli Grsec. Patrně propustnost disku. To, že test neprokázal odlišnosti nevylučuje, že by se při použití rychlejšího disku – např. SSD – situace nezměnila a Grsec se neukázal býti brzdou datové propustnosti.

13.8.3 Test sysbench_fileio_rndwr_s

Test 13.8.3 je odvozen od 13.8.2. Liší se od něj v tom, že tentokrát nad danými soubory neprobíhá náhodné čtení, ale náhodný zápis.

NZPV (x)	velikost bloku
jednotka NZPV	B
ZPV (y)	propustnost
jednotka ZPV	Mb/s
počet opakování měření	3
počítač spouštějící testovací příkaz	xenmv

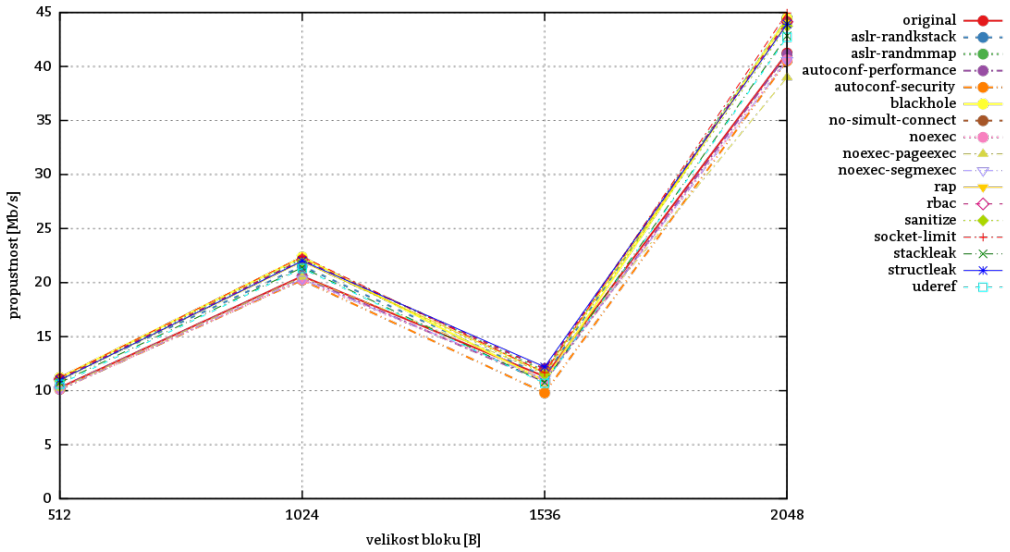
testovací příkaz	`sysbench --test=fileio --file-num=100 --file-total-size=200K`
	`↪ --max-requests=100000 --max-time=200`
	`↪ --file-extra-flags=direct --file-block-size=${X}`
	`↪ --file-test-mode=rndwr run`

Tabulka 13.79: Charakteristiky testu sysbench_fileio_rndwr_s

13.8.3.1 Naměřené hodnoty pro jednotlivé konfigurace

měřená konfigurace	velikost bloku [B] 512	1024	1536	2048
original	10,281	20,592	11,29	41,24
aslr-randkstack	11,118	21,523	11,31	44,46
aslr-randmmap	11,127	22,202	11,82	44,39
autoconf-performance	10,279	20,505	10,82	41,13
autoconf-security	10,128	20,226	9,79	40,52
blackhole	11,214	22,367	10,88	44,45
no-simult-connect	11,09	22,119	12,02	44,14
noexec	10,127	20,278	11,00	40,62
noexec-pageexec	10,214	20,508	11,69	38,99
noexec-segmexec	10,089	20,323	11,05	40,56
rap	11,004	21,94	11,68	44,04
rbac	11,066	22,095	11,99	44,13
sanitize	10,921	21,968	11,51	43,73
socket-limit	11,223	22,403	11,66	44,94
stackleak	10,745	21,388	10,76	42,83
structleak	10,989	22,007	12,19	43,91
uderef	10,6	21,307	10,68	42,70

Tabulka 13.80: Absolutní srovnání konfigurací

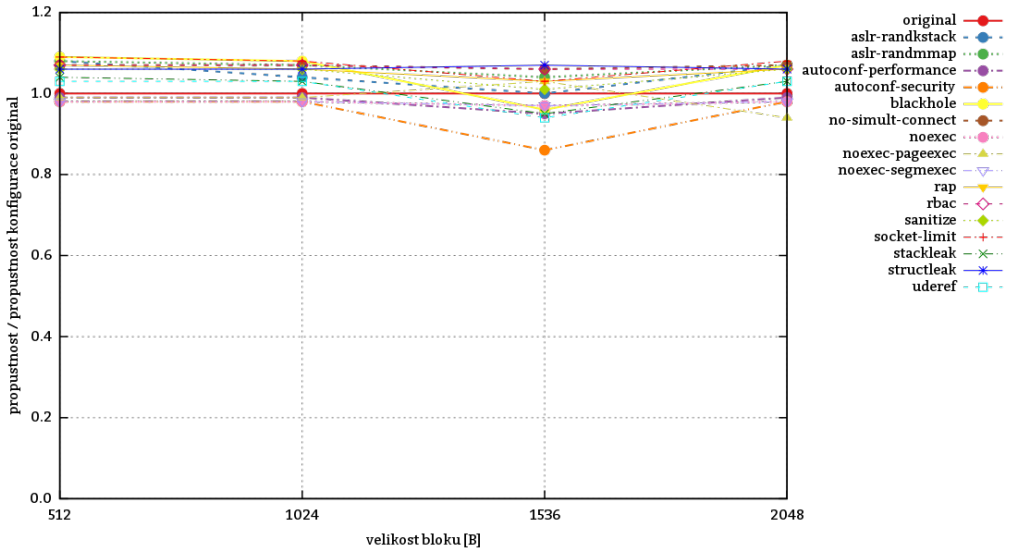


Obrázek 13.20: Absolutní srovnání konfigurací

13.8.3.2 Srovnání konfigurací

měřená konfigurace	velikost bloku [B] 512	1024	1536	2048	průměr
original	1	1	1	1	1
aslr-randkstack	1,08	1,04	1	1,07	1,04
aslr-randmmap	1,08	1,07	1,04	1,07	1,06
autoconf-performance	0,99	0,99	0,95	0,99	0,98
autoconf-security	0,98	0,98	0,86	0,98	0,95
blackhole	1,09	1,08	0,96	1,07	1,05
no-simult-connect	1,07	1,07	1,06	1,07	1,06
noexec	0,98	0,98	0,97	0,98	0,97
noexec-pageexec	0,99	0,99	1,03	0,94	0,98
noexec-segmexec	0,98	0,98	0,97	0,98	0,97
rap	1,07	1,06	1,03	1,06	1,05
rbac	1,07	1,07	1,06	1,06	1,06
sanitize	1,06	1,06	1,01	1,06	1,04
socket-limit	1,09	1,08	1,03	1,08	1,07
stackleak	1,04	1,03	0,95	1,03	1,01
structleak	1,06	1,06	1,07	1,06	1,06
uderef	1,03	1,03	0,94	1,03	1

Tabulka 13.81: Relativní srovnání konfigurací vůči referenční konfiguraci original



Obrázek 13.21: Relativní srovnání konfigurací vůči referenční konfiguraci original

13.8.3.3 Zhodnocení testu

Naměřené absolutní hodnoty jsou nižší, nežli v testu 13.8.2. Výsledek srovnání jednotlivých konfigurací je však stejný. Test ukázal, že na propustnost při náhodném zápisu velkého množství malých souborů nemá Grsec při použití daného úložiště vliv čili že úzkým hrdlem se v tomto případě stala patrně propustnost disku samotného.

13.8.4 Test sysbench_fileio_rndwr_l

Test sysbench_fileio_rndwr_l testuje propustnost při náhodném zápisu do 5 přibližně stejně velkých souborů s celkovou velikostí 1 GiB. Přepínačem `--file-extra-flags=direct` jsou deaktivovány mezipaměti, které je program deaktivovat schopen.

13.8.4.1 Naměřené hodnoty pro jednotlivé konfigurace

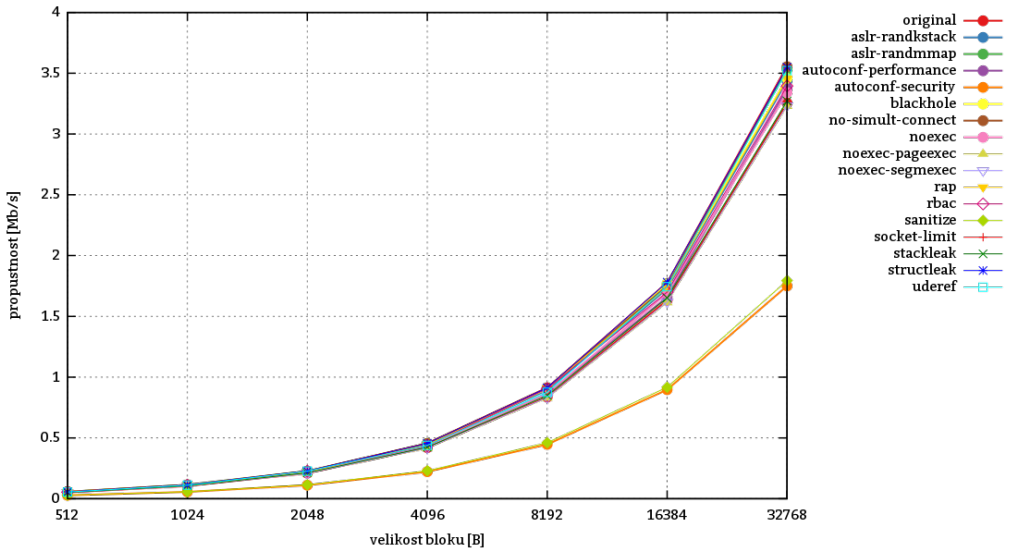
NZPV (x)	velikost bloku
jednotka NZPV	B
ZPV (y)	propustnost
jednotka ZPV	Mb/s
počet opakování měření	3
počítač spouštějící testovací příkaz	xenmv

testovací příkaz	`sysbench --test=fileio --file-num=5 --file-total-size=1G`
	`↪ --max-requests=100000 --max-time=200`
	`↪ --file-extra-flags=direct --file-block-size=${X}`
	`↪ --file-test-mode=rndwr run`

Tabulka 13.82: Charakteristiky testu sysbench_fileio_rndwr_l

měřená konfigurace	**velikost bloku [B]**						
	512	**1024**	**2048**	**4096**	**8192**	**16384**	**32768**
original	0,052	0,105	0,21	0,42	0,836	1,633	3,254
aslr-randkstack	0,056	0,111	0,225	0,447	0,893	1,727	3,437
aslr-randmmap	0,057	0,114	0,227	0,455	0,91	1,769	3,531
autoconf-performance	0,054	0,108	0,216	0,433	0,863	1,659	3,374
autoconf-security	0,028	0,056	0,111	0,222	0,447	0,897	1,75
blackhole	0,056	0,111	0,221	0,444	0,905	1,769	3,458
no-simult-connect	0,057	0,114	0,227	0,454	0,913	1,751	3,554
noexec	0,054	0,108	0,215	0,433	0,865	1,69	3,342
noexec-pageexec	0,052	0,104	0,208	0,415	0,829	1,614	3,229
noexec-segmexec	0,052	0,104	0,208	0,417	0,832	1,618	3,233
rap	0,057	0,113	0,22	0,444	0,911	1,735	3,457
rbac	0,054	0,112	0,227	0,436	0,906	1,7	3,395
sanitize	0,029	0,057	0,115	0,23	0,461	0,917	1,794
socket-limit	0,057	0,114	0,228	0,459	0,915	1,783	3,566
stackleak	0,052	0,106	0,213	0,424	0,849	1,652	3,271
structleak	0,056	0,114	0,228	0,456	0,913	1,781	3,55
uderef	0,055	0,112	0,227	0,439	0,876	1,744	3,522

Tabulka 13.83: Absolutní srovnání konfigurací

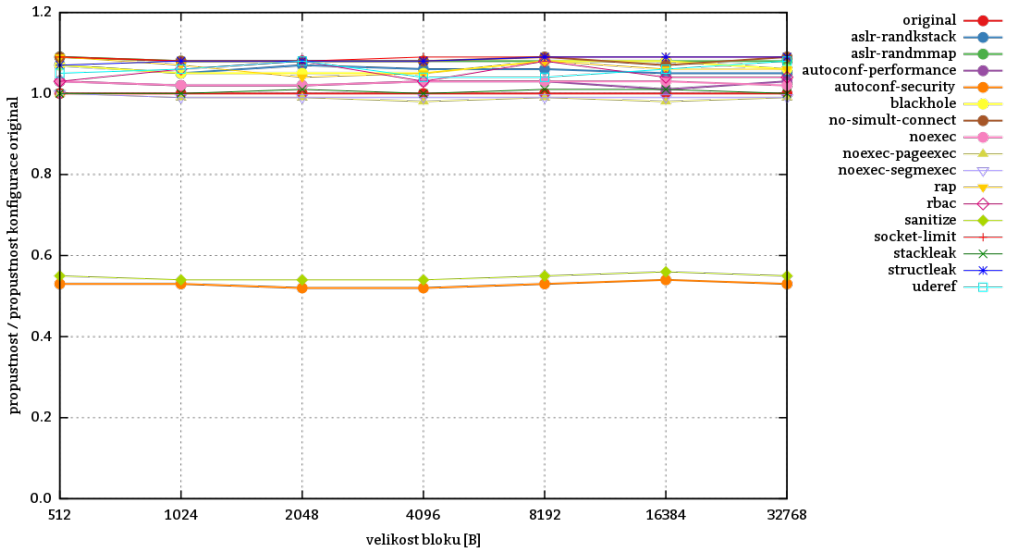


Obrázek 13.22: Absolutní srovnání konfigurací

13.8.4.2 Srovnání konfigurací

měřená konfigurace	velikost bloku [B] 512	1024	2048	4096	8192	16384	32768	průměr
original	1	1	1	1	1	1	1	1
aslr-randkstack	1,07	1,05	1,07	1,06	1,06	1,05	1,05	1,05
aslr-randmmap	1,09	1,08	1,08	1,08	1,08	1,08	1,08	1,08
autoconf-performance	1,03	1,02	1,02	1,03	1,03	1,01	1,03	1,02
autoconf-security	0,53	0,53	0,52	0,52	0,53	0,54	0,53	0,52
blackhole	1,07	1,05	1,05	1,05	1,08	1,08	1,06	1,06
no-simult-connect	1,09	1,08	1,08	1,08	1,09	1,07	1,09	1,08
noexec	1,03	1,02	1,02	1,03	1,03	1,03	1,02	1,02
noexec-pageexec	1	0,99	0,99	0,98	0,99	0,98	0,99	0,98
noexec-segmexec	1	0,99	0,99	0,99	0,99	0,99	0,99	0,99
rap	1,09	1,07	1,04	1,05	1,08	1,06	1,06	1,06
rbac	1,03	1,06	1,08	1,03	1,08	1,04	1,04	1,05
sanitize	0,55	0,54	0,54	0,54	0,55	0,56	0,55	0,54
socket-limit	1,09	1,08	1,08	1,09	1,09	1,09	1,09	1,08
stackleak	1	1	1,01	1	1,01	1,01	1	1
structleak	1,07	1,08	1,08	1,08	1,09	1,09	1,09	1,08
uderef	1,05	1,06	1,08	1,04	1,04	1,06	1,08	1,05

Tabulka 13.84: Relativní srovnání konfigurací vůči referenční konfiguraci original



Obrázek 13.23: Relativní srovnání konfigurací vůči referenční konfiguraci original

13.8.4.3 Zhodnocení testu

Propustnost při náhodném zápisu do velkých souborů s patchem Grsec zůstává v blízkosti propustnosti systému v referenční konfiguraci. Zásadní zpomalení je pozorovatelné u systému v konfiguracích sanitize a autoconf-security. Za pokles propustnosti je odpovědná volba `PAX_MEMORY_SANITIZE`, která je obsažena v obou zmíněných konfiguracích.

Testovací nástroj alokuje prostor v hlavní paměti, kterého se následně zbavuje – dealokuje jej. V tu chvíli musí zakročit jádro s aktivovanou volbou `PAX_MEMORY_SANITIZE`, které obsah této paměti smaže, čímž dosáhne toho, že se obsah ležící na dané paměti stane pro jakýkoliv další proces, který se k ní dostane, bezcenným.

13.8.5 Test sysbench_fileio_seqrd_1

Test sysbench_fileio_seqrd_1 testuje propustnost při sekvenčním čtení 5 přibližně stejně velkých souborů s celkovou velikostí 1 GiB. Přepínačem `--file-extra-flags=direct` jsou deaktivovány mezipaměti, které je program deaktivovat schopen.

NZPV (x)	velikost bloku
jednotka NZPV	B
ZPV (y)	propustnost
jednotka ZPV	Mb/s
počet opakování měření	3
počítač spouštějící testovací příkaz	xenmv

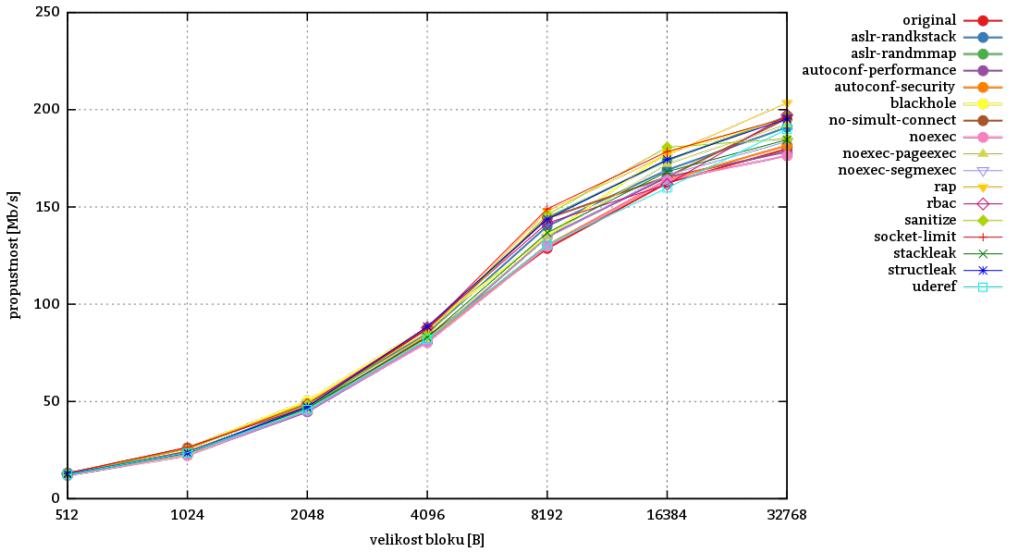
testovací příkaz	`sysbench --test=fileio --file-num=5 --file-total-size=1G`
	`↪ --max-requests=100000 --max-time=200`
	`↪ --file-extra-flags=direct --file-block-size=${X}`
	`↪ --file-test-mode=seqrd run`

Tabulka 13.85: Charakteristiky testu sysbench_fileio_seqrd_1

13.8.5.1 Naměřené hodnoty pro jednotlivé konfigurace

měřená konfigurace	velikost bloku [B] 512	1024	2048	4096	8192	16384	32768
original	12,435	24,216	45,86	83,384	128,69	162,4	179,8
aslr-randkstack	13,074	23,606	47,49	85,172	140,20	168,94	190,9
aslr-randmmap	13,069	23,537	46,18	87,575	144,43	173,96	196,5
autoconf-performance	12,164	22,528	44,68	80,812	134,71	165,36	178,4
autoconf-security	12,195	22,506	45,56	81,533	130,30	163,91	181,6
blackhole	12,905	26,036	50,23	88,118	135,15	177,98	195,6
no-simult-connect	12,96	26,099	48,69	87,227	144,12	165,44	195,5
noexec	12,035	22,173	45,26	80,321	129,65	163,75	176,3
noexec-pageexec	12,144	22,477	45,86	82,038	134,80	172,05	192,3
noexec-segmexec	11,979	23,07	45,05	81,376	134,16	166,37	183,6
rap	12,846	26,029	49,60	85,528	147,96	176,8	203,4
rbac	12,723	25,645	48,45	87,94	141,7	161,61	197,2
sanitize	12,9	25,509	48,33	84,046	146,06	180,69	185,0
socket-limit	13,13	26,373	47,36	87,351	148,98	178,47	195,8
stackleak	12,428	23,555	46,61	82,968	136,72	168,14	184,5
structleak	12,865	23,557	47,21	88,554	143,72	174,41	195,3
uderef	12,493	23,687	45,82	81,995	130,57	159,80	190,0

Tabulka 13.86: Absolutní srovnání konfigurací

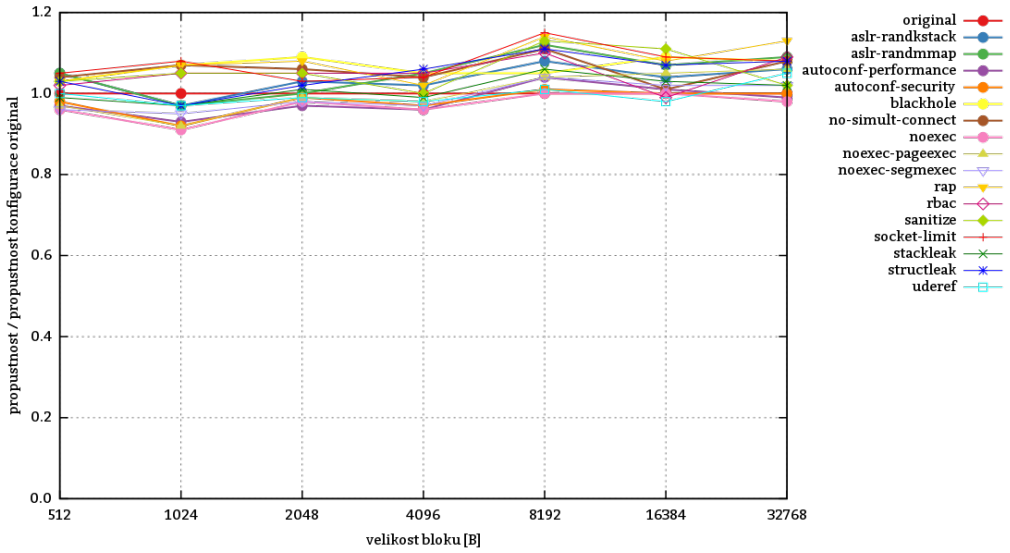


Obrázek 13.24: Absolutní srovnání konfigurací

13.8.5.2 Srovnání konfigurací

měřená konfigurace	velikost bloku [B] 512	1024	2048	4096	8192	16384	32768	průměr
original	1	1	1	1	1	1	1	1
aslr-randkstack	1,05	0,97	1,03	1,02	1,08	1,04	1,06	1,03
aslr-randmmap	1,05	0,97	1	1,05	1,12	1,07	1,09	1,05
autoconf-performance	0,97	0,93	0,97	0,96	1,04	1,01	0,99	0,98
autoconf-security	0,98	0,92	0,99	0,97	1,01	1	1	0,98
blackhole	1,03	1,07	1,09	1,05	1,05	1,09	1,08	1,06
no-simult-connect	1,04	1,07	1,06	1,04	1,11	1,01	1,08	1,05
noexec	0,96	0,91	0,98	0,96	1	1	0,98	0,97
noexec-pageexec	0,97	0,92	0,99	0,98	1,04	1,05	1,06	1
noexec-segmexec	0,96	0,95	0,98	0,97	1,04	1,02	1,02	0,99
rap	1,03	1,07	1,08	1,02	1,14	1,08	1,13	1,07
rbac	1,02	1,05	1,05	1,05	1,1	0,99	1,09	1,05
sanitize	1,03	1,05	1,05	1	1,13	1,11	1,02	1,05
socket-limit	1,05	1,08	1,03	1,04	1,15	1,09	1,08	1,07
stackleak	0,99	0,97	1,01	0,99	1,06	1,03	1,02	1,01
structleak	1,03	0,97	1,02	1,06	1,11	1,07	1,08	1,04
uderef	1	0,97	0,99	0,98	1,01	0,98	1,05	0,99

Tabulka 13.87: Relativní srovnání konfigurací vůči referenční konfiguraci original



Obrázek 13.25: Relativní srovnání konfigurací vůči referenční konfiguraci original

13.8.5.3 Zhodnocení testu

Propustnost při sekvenčním čtení velkých souborů není patchem Grsec příliš dotčena.

13.8.6 Test sysbench_fileio_seqwr_l

Test sysbench_fileio_seqwr_l testuje propustnost při sekvenčním zápisu do 5 přibližně stejně velkých souborů s celkovou velikostí 1 GiB. Přepínačem `--file-extra-flags=direct` jsou deaktivovány mezipaměti, které je program deaktivovat schopen.

13.8.6.1 Naměřené hodnoty pro jednotlivé konfigurace

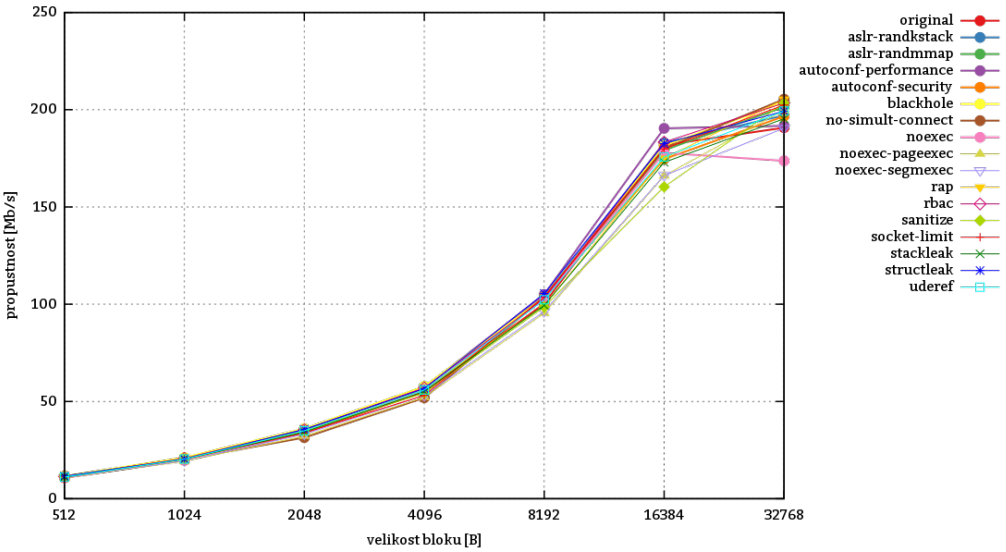
NZPV (x)	velikost bloku
jednotka NZPV	B
ZPV (y)	propustnost
jednotka ZPV	Mb/s
počet opakování měření	3
počítač spouštějící testovací příkaz	xenmv

testovací příkaz	`sysbench --test=fileio --file-num=5 --file-total-size=1G`
	`↪ --max-requests=100000 --max-time=200`
	`↪ --file-extra-flags=direct --file-block-size=${X}`
	`↪ --file-test-mode=seqwr run`

Tabulka 13.88: Charakteristiky testu sysbench_fileio_seqwr_l

měřená konfigurace	velikost bloku [B] 512	1024	2048	4096	8192	16384	32768
original	10,818	19,79	33,872	55,013	100,3	181,8	190,7
aslr-randkstack	11,479	20,68	35,579	57,302	103,8	182,95	197,0
aslr-randmmap	11,517	20,86	35,443	57,618	102,7	179,21	201,4
autoconf-performance	11,061	20,28	35,329	57,197	105,	190,4	191,6
autoconf-security	10,917	19,92	34,339	55,328	103,5	174,60	196,6
blackhole	11,598	21,17	36,026	57,678	103,5	181,57	203,8
no-simult-connect	11,355	20,29	31,448	51,865	103,6	180,07	205,3
noexec	10,995	19,91	34,281	55,86	102,7	177,76	173,7
noexec-pageexec	10,663	19,27	32,204	52,194	95,3	166,02	199,8
noexec-segmexec	10,737	19,50	33,154	53,009	96,2	166,20	190,4
rap	11,396	20,06	34,412	56,319	99,3	175,02	205,2
rbac	11,397	20,83	35,589	57,017	102,7	183,31	203,4
sanitize	11,39	20,63	34,719	54,329	98,6	160,33	203,5
socket-limit	11,54	20,58	33,848	53,145	103,6	179,87	202,4
stackleak	11,06	20,33	34,28	55,097	99,4	172,97	195,3
structleak	11,634	20,41	35,721	56,605	105,5	182,6	199,2
uderef	11,192	20,41	34,943	55,945	102,3	175,70	199,3

Tabulka 13.89: Absolutní srovnání konfigurací

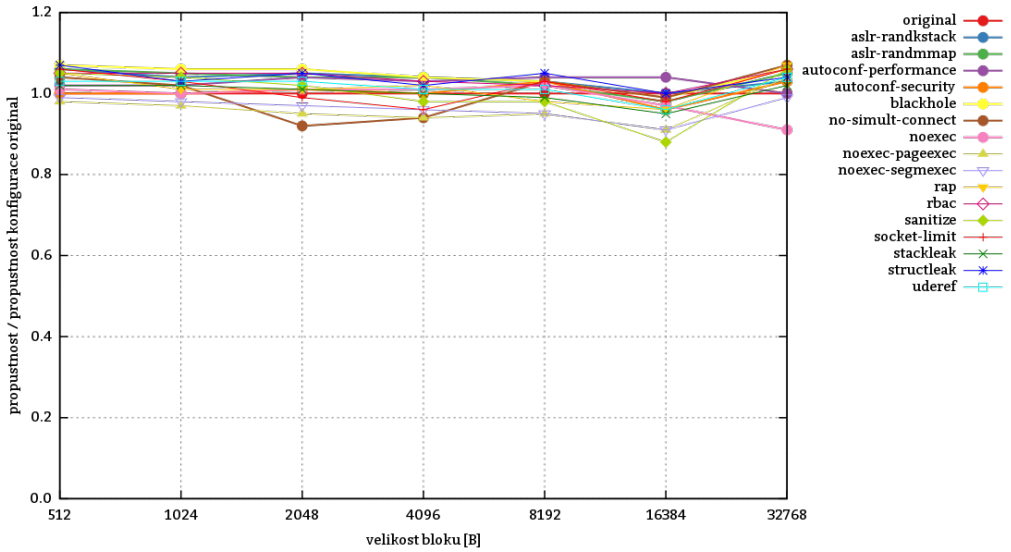


Obrázek 13.26: Absolutní srovnání konfigurací

13.8.6.2 Srovnání konfigurací

měřená konfigurace	velikost bloku [B] 512	1024	2048	4096	8192	16384	32768	průměr
original	1	1	1	1	1	1	1	1
aslr-randkstack	1,06	1,04	1,05	1,04	1,03	1	1,03	1,03
aslr-randmmap	1,06	1,05	1,04	1,04	1,02	0,98	1,05	1,03
autoconf-performance	1,02	1,02	1,04	1,03	1,04	1,04	1	1,02
autoconf-security	1	1	1,01	1	1,03	0,96	1,03	1
blackhole	1,07	1,06	1,06	1,04	1,03	0,99	1,06	1,04
no-simult-connect	1,04	1,02	0,92	0,94	1,03	0,99	1,07	1
noexec	1,01	1	1,01	1,01	1,02	0,97	0,91	0,99
noexec-pageexec	0,98	0,97	0,95	0,94	0,95	0,91	1,04	0,96
noexec-segmexec	0,99	0,98	0,97	0,96	0,95	0,91	0,99	0,96
rap	1,05	1,01	1,01	1,02	0,98	0,96	1,07	1,01
rbac	1,05	1,05	1,05	1,03	1,02	1	1,06	1,03
sanitize	1,05	1,04	1,02	0,98	0,98	0,88	1,06	1
socket-limit	1,06	1,03	0,99	0,96	1,03	0,98	1,06	1,01
stackleak	1,02	1,02	1,01	1	0,99	0,95	1,02	1
structleak	1,07	1,03	1,05	1,02	1,05	1	1,04	1,03
uderef	1,03	1,03	1,03	1,01	1,01	0,96	1,04	1,01

Tabulka 13.90: Relativní srovnání konfigurací vůči referenční konfiguraci original



Obrázek 13.27: Relativní srovnání konfigurací vůči referenční konfiguraci original

13.8.6.3 Zhodnocení testu

Test ukázal, že propustnost při sekvenčním zápisu do několika velkých souborů není patchem Grsec příliš dotčena.

13.8.7 Test sysbench_memory_write_seq

Test `sysbench_memory_write_seq` zjišťuje propustnost hlavní paměti při sekvenčním zápisu 50 GB dat. Tento zápis je prováděn po blocích a právě velikost bloku vstupuje do testu jako NZPV. Zápis provádí jedno vlákno. Systém, na němž testování probíhalo disponuje 16,796 GB hlavní paměti. Test v hlavní paměti alokuje jemu předepsanou velikost bloku, do které následně zapisuje. Největší velikost bloku v testu – 10000000 kB – zabírá 0,595 hlavní paměti testovaného systému.

NZPV (x)	velikost manipulované oblasti
jednotka NZPV	KB
ZPV (y)	propustnost
jednotka ZPV	GB/s
počet opakování měření	3
počítač spouštějící testovací příkaz	xenmv

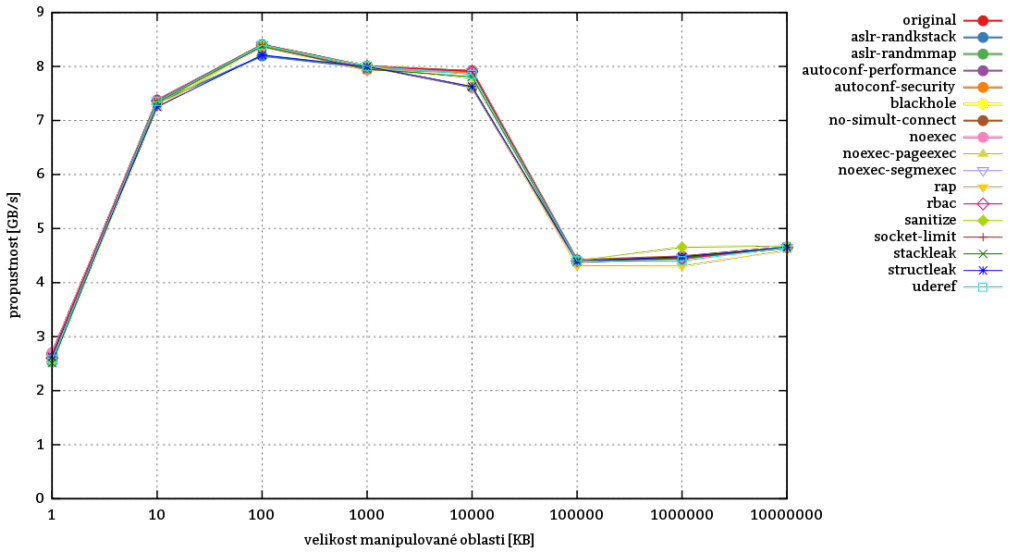
testovací příkaz	`sysbench --test=memory --num-threads=1 --memory-oper=write`
	`↪ --memory-access-mode=seq --memory-block-size=${X}K`
	`↪ --memory-total-size=50G run`

Tabulka 13.91: Charakteristiky testu sysbench_memory_write_seq

13.8.7.1 Naměřené hodnoty pro jednotlivé konfigurace

měřená konfigurace	velikost manipulované oblasti [KB] 1	10	100	1000	10000	100000	1000000	10000000
original	2,695	7,332	8,398	8,004	7,912	4,382	4,451	4,65
aslr-randkstack	2,534	7,329	8,196	7,979	7,898	4,397	4,468	4,656
aslr-randmmap	2,68	7,366	8,38	8,006	7,914	4,395	4,459	4,655
autoconf-performance	2,664	7,378	8,395	8,004	7,614	4,385	4,455	4,652
autoconf-security	2,613	7,362	8,374	7,941	7,89	4,396	4,438	4,653
blackhole	2,672	7,364	8,384	8,008	7,912	4,383	4,487	4,658
no-simult-connect	2,672	7,366	8,403	7,979	7,919	4,421	4,486	4,656
noexec	2,695	7,36	8,396	7,998	7,91	4,394	4,472	4,655
noexec-pageexec	2,632	7,238	8,387	7,99	7,783	4,394	4,477	4,657
noexec-segmexec	2,597	7,367	8,402	7,951	7,902	4,398	4,471	4,655
rap	2,539	7,252	8,405	7,991	7,633	4,311	4,305	4,599
rbac	2,605	7,367	8,394	8,005	7,91	4,403	4,443	4,653
sanitize	2,525	7,308	8,396	8,004	7,618	4,409	4,655	4,679
socket-limit	2,673	7,367	8,406	8,006	7,912	4,404	4,477	4,647
stackleak	2,511	7,344	8,371	7,951	7,808	4,39	4,459	4,655
structleak	2,622	7,252	8,212	7,997	7,629	4,392	4,48	4,656
uderef	2,581	7,354	8,398	8,005	7,851	4,399	4,403	4,649

Tabulka 13.92: Absolutní srovnání konfigurací

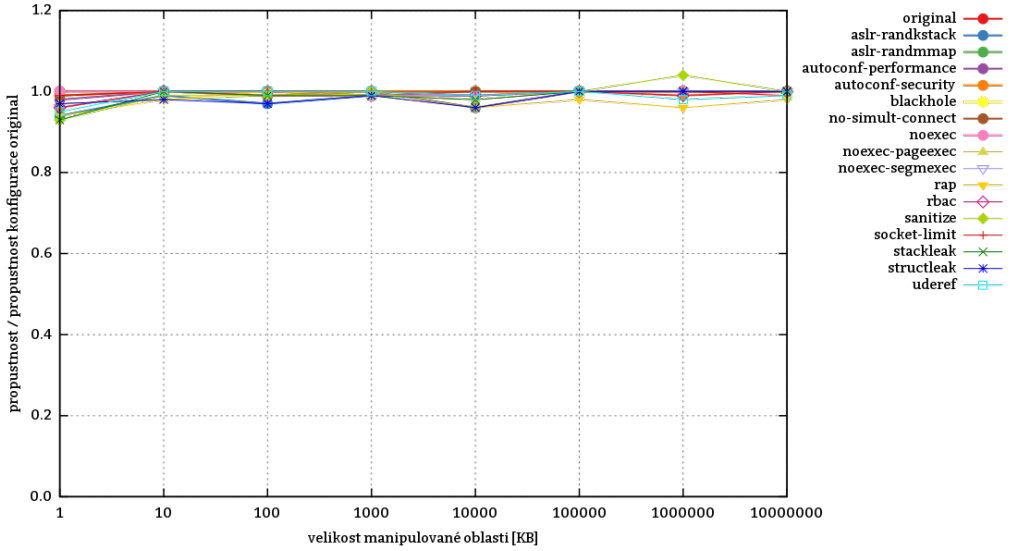


Obrázek 13.28: Absolutní srovnání konfigurací

13.8.7.2 Srovnání konfigurací

měřená konfigurace	velikost manipulované oblasti [KB] 1	10	100	1000	10000	100000	1000000	10000000	průměr
original	1	1	1	1	1	1	1	1	1
aslr-randkstack	0,94	0,99	0,97	0,99	0,99	1	1	1	0,98
aslr-randmmap	0,99	1	0,99	1	1	1	1	1	0,99
autoconf-performance	0,98	1	0,99	1	0,96	1	1	1	0,99
autoconf-security	0,96	1	0,99	0,99	0,99	1	0,99	1	0,99
blackhole	0,99	1	0,99	1	1	1	1	1	0,99
no-simult-connect	0,99	1	1	0,99	1	1	1	1	0,99
noexec	1	1	0,99	0,99	0,99	1	1	1	0,99
noexec-pageexec	0,97	0,98	0,99	0,99	0,98	1	1	1	0,98
noexec-segmexec	0,96	1	1	0,99	0,99	1	1	1	0,99
rap	0,94	0,98	1	0,99	0,96	0,98	0,96	0,98	0,97
rbac	0,96	1	0,99	1	0,99	1	0,99	1	0,99
sanitize	0,93	0,99	0,99	1	0,96	1	1,04	1	0,98
socket-limit	0,99	1	1	1	1	1	1	0,99	0,99
stackleak	0,93	1	0,99	0,99	0,98	1	1	1	0,98
structleak	0,97	0,98	0,97	0,99	0,96	1	1	1	0,98
uderef	0,95	1	1	1	0,99	1	0,98	0,99	0,98

Tabulka 13.93: Relativní srovnání konfigurací vůči referenční konfiguraci original



Obrázek 13.29: Relativní srovnání konfigurací vůči referenční konfiguraci original

13.8.7.3 Zhodnocení testu

Propustnost hlavní paměti při sekvenčním zápisu jednoho jejího úseku je nehledě na konfiguraci testovaného systému optimální při velikost bloku 100 kB. Srovnatelná pak v intervalu 10 kB až 10000 kB. Při ostatních velikostech bloků je práce s hlavní pamětí neefektivní, avšak možná.

13.8.8 Test sysbench_memory_write_seq

Test `sysbench_memory_write_seq` zjišťuje propustnost paměti při sekvenčním zápisu 50 GB dat po blocích velikosti 1 kB.

13.8.8.1 Naměřené hodnoty pro jednotlivé konfigurace

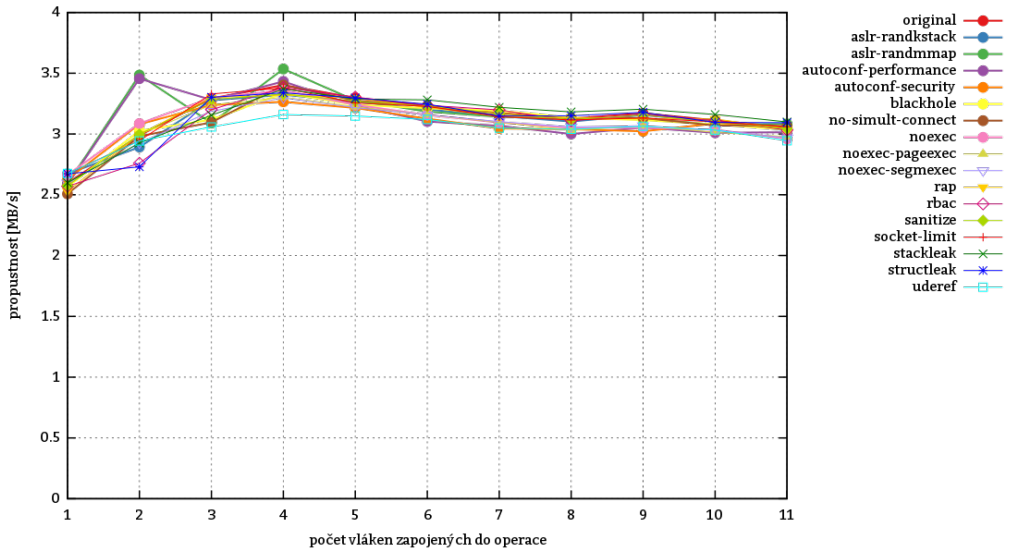
NZPV (x)	počet vláken zapojených do operace
jednotka NZPV	není (bezrozměrná veličina)
ZPV (y)	propustnost
jednotka ZPV	MB/s
počet opakování měření	3
počítač spouštějící testovací příkaz	xenmv

testovací příkaz	`sysbench --test=memory --num-threads=${X} --memory-oper=write`
	`↪ --memory-access-mode=seq --memory-block-size=1K`
	`↪ --memory-total-size=50G run`

Tabulka 13.94: Charakteristiky testu sysbench_memory_write_seq

měřená konfigurace	počet vláken zapojených do operace										
	1	2	3	4	5	6	7	8	9	10	11
original	2,596	2,989	3,293	3,403	3,297	3,231	3,174	3,127	3,17	3,113	3,054
aslr-randkstack	2,672	2,893	3,278	3,323	3,266	3,202	3,147	3,103	3,156	3,102	3,03
aslr-randmmap	2,589	3,485	3,09	3,535	3,276	3,184	3,142	3,126	3,129	3,076	3,083
autoconf-performance	2,583	3,455	3,281	3,432	3,235	3,103	3,067	3,	3,053	3,01	3,015
autoconf-security	2,627	3,081	3,238	3,265	3,215	3,126	3,046	3,04	3,022	3,086	3,047
blackhole	2,565	3,026	3,118	3,336	3,241	3,212	3,192	3,135	3,114	3,079	3,036
no-simult-connect	2,508	2,982	3,096	3,402	3,275	3,229	3,152	3,129	3,131	3,07	3,066
noexec	2,664	3,086	3,296	3,359	3,24	3,156	3,095	3,051	3,052	3,035	2,96
noexec-pageexec	2,577	2,997	3,24	3,303	3,225	3,157	3,097	3,031	3,068	3,014	2,978
noexec-segmexec	2,597	2,926	3,187	3,297	3,217	3,158	3,102	3,058	3,068	3,033	2,969
rap	2,53	2,998	3,301	3,32	3,296	3,218	3,144	3,134	3,125	3,124	3,041
rbac	2,57	2,76	3,2	3,382	3,304	3,236	3,198	3,109	3,156	3,109	3,033
sanitize	2,575	2,998	3,283	3,33	3,27	3,227	3,171	3,119	3,154	3,1	3,044
socket-limit	2,674	2,946	3,33	3,378	3,256	3,227	3,142	3,116	3,132	3,1	3,06
stackleak	2,599	2,926	3,155	3,371	3,288	3,281	3,219	3,182	3,204	3,161	3,099
structleak	2,673	2,73	3,302	3,339	3,297	3,245	3,148	3,15	3,178	3,095	3,093
uderef	2,675	2,943	3,058	3,16	3,148	3,12	3,049	3,039	3,064	3,033	2,945

Tabulka 13.95: Absolutní srovnání konfigurací

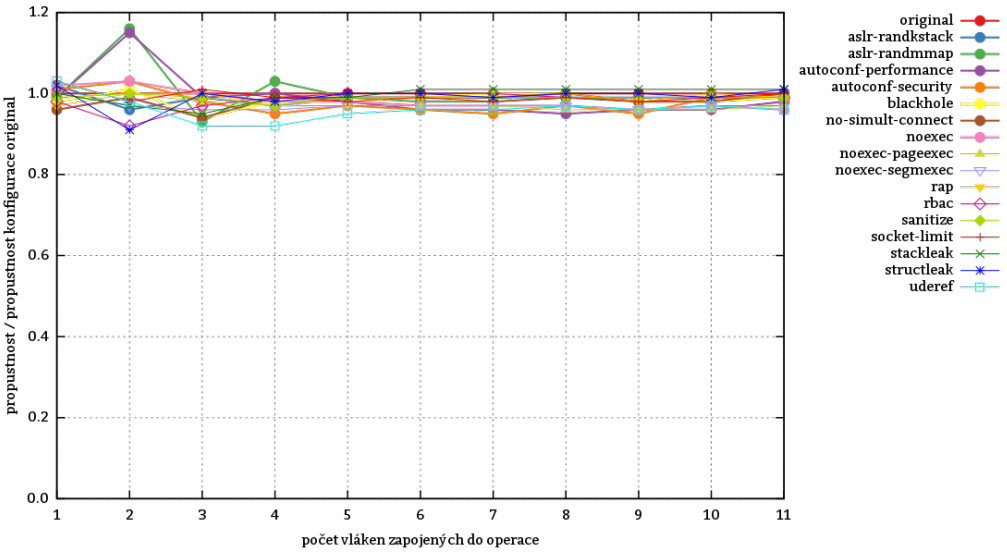


Obrázek 13.30: Absolutní srovnání konfigurací

13.8.8.2 Srovnání konfigurací

měřená konfigurace	počet vláken zapojených do operace 1	2	3	4	5	6	7	8	9	10	11	průměr
original	1	1	1	1	1	1	1	1	1	1	1	1
aslr-randkstack	1,02	0,96	0,99	0,97	0,99	0,99	0,99	0,99	0,99	0,99	0,99	0,98
aslr-randmmap	0,99	1,16	0,93	1,03	0,99	0,98	0,98	0,99	0,98	0,98	1	1
autoconf-performance	0,99	1,15	0,99	1	0,98	0,96	0,96	0,95	0,96	0,96	0,98	0,98
autoconf-security	1,01	1,03	0,98	0,95	0,97	0,96	0,95	0,97	0,95	0,99	0,99	0,97
blackhole	0,98	1,01	0,94	0,98	0,98	0,99	1	1	0,98	0,98	0,99	0,98
no-simult-connect	0,96	0,99	0,94	0,99	0,99	0,99	0,99	1	0,98	0,98	1	0,98
noexec	1,02	1,03	1	0,98	0,98	0,97	0,97	0,97	0,96	0,97	0,96	0,98
noexec-pageexec	0,99	1	0,98	0,97	0,97	0,97	0,97	0,96	0,96	0,96	0,97	0,97
noexec-segmexec	1	0,97	0,96	0,96	0,97	0,97	0,97	0,97	0,96	0,97	0,97	0,97
rap	0,97	1	1	0,97	0,99	0,99	0,99	1	0,98	1	0,99	0,98
rbac	0,98	0,92	0,97	0,99	1	1	1	0,99	0,99	0,99	0,99	0,98
sanitize	0,99	1	0,99	0,97	0,99	0,99	0,99	0,99	0,99	0,99	0,99	0,98
socket-limit	1,03	0,98	1,01	0,99	0,98	0,99	0,98	0,99	0,98	0,99	1	0,99
stackleak	1	0,97	0,95	0,99	0,99	1,01	1,01	1,01	1,01	1,01	1,01	0,99
structleak	1,02	0,91	1	0,98	1	1	0,99	1	1	0,99	1,01	0,99
udereference	1,03	0,98	0,92	0,92	0,95	0,96	0,96	0,97	0,96	0,97	0,96	0,96

Tabulka 13.96: Relativní srovnání konfigurací vůči referenční konfiguraci original



Obrázek 13.31: Relativní srovnání konfigurací vůči referenční konfiguraci original

13.8.8.3 Zhodnocení testu

Test ukázal, že nasazení Grsec ve zkoumané disciplíně příliš nemá vliv.

13.8.9 Test sysbench_db_readonly

Test `sysbench_db_readonly` měří výkon databáze MySQL, k čemuž je potřeba především její přítomnost. Proto byla na xenmv pro účely měření doinstalována a byla v ní vytvořena databáze:

```
aptitude install mysql-server
mysql -u root -p -e 'create database db;'
```

Test vytvoří tabulku odpovídající dotazu

```
CREATE TABLE `sbtest` (
    `id` int(10) unsigned NOT NULL auto_increment,
    `k` int(10) unsigned NOT NULL default '0',
    `c` char(120) NOT NULL default '',
    `pad` char(60) NOT NULL default '',
    PRIMARY KEY (`id`),
    KEY `k` (`k`);
)
```

Nad touto tabulkou následně provádí čtení a zápis. Význam jednotlivých přepínačů testovacího příkazu je popsán v tabulce 13.97. Na databázi během testu směřují **výhradně SELECT dotazy**. Z databáze se tak data pouze čtou. Zamě-

Přepínač	Význam
`--oltp-table-size`	Počet řádků tabulky.
`--num-threads`	Počet vláken přistupujících k tabulce.
`--max-requests`	Maximální počet požadavků. Hodnota 0 značí „neomezený“.
`--max-time`	Maximální trvání testu v sekundách.

Tabulka 13.97: Parametry příkazu `sysbench` v měření `sysbench_db`

ření testu na čtení vylučuje použití transakcí, které proto byly deaktivovány (přepínačem `--oltp-skip-trx`). Testovací nástroj s databází komunikuje prostřednictvím socketu mapovaného do souboru `/var/run/mysqld/mysqld.sock`.

13.8.9.1 Naměřené hodnoty pro jednotlivé konfigurace

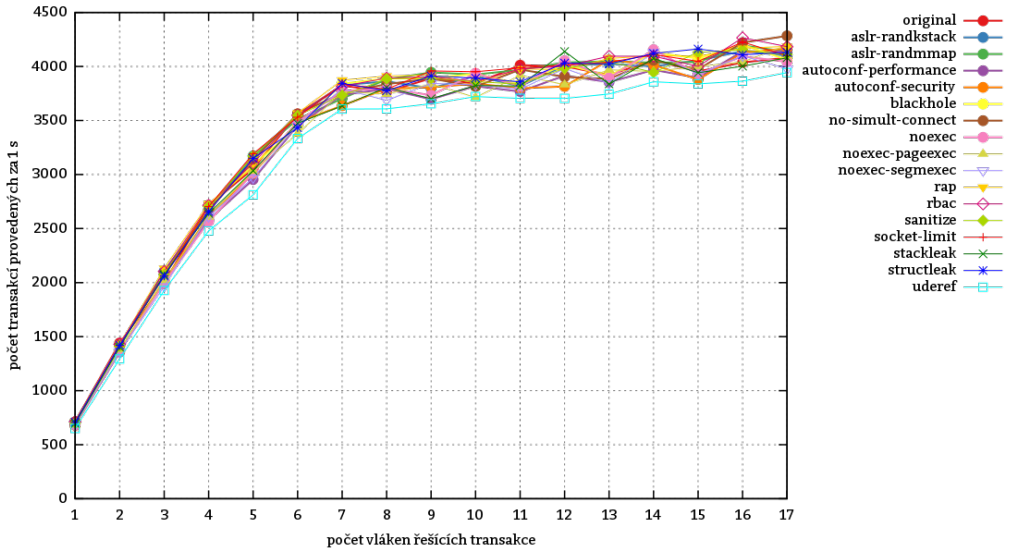
NZPV (x)	počet vláken řešících transakce
jednotka NZPV	není (bezrozměrná veličina)
ZPV (y)	počet transakcí provedených za 1 s
jednotka ZPV	není (bezrozměrná veličina)
počet opakování měření	1
počítač spouštějící testovací příkaz	xenmv

testovací příkaz	`sysbench --test=oltp --mysql-db=db --mysql-user=root`
	`↪ --mysql-password=+ --oltp-table-size=1000000 --max-requests=0`
	`↪ --db-driver=mysql --oltp-read-only=on --oltp-skip-trx`
	`↪ --oltp-nontrx-mode=select --max-time=20 --num-threads=${X}`
	`↪ run`

Tabulka 13.98: Charakteristiky testu sysbench_db_readonly

měřená konfigurace	počet vláken řešících transakce 1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
original	712,6	1440,8	1991	2715,5	3052	3561,0	3823,9	3763,9	3891,4	3843,	4011	4001	3920	4057	3953	4037	4080
aslr-randkstack	693,2	1417,6	2105	2673,4	3170	3544,0	3827,0	3862,3	3798,0	3912,0	3817	4007	4024	4000	4054	4142	4100
aslr-randmmap	711,	1423,4	2077	2680,2	3169	3509,8	3703,7	3881,0	3945,1	3925,4	3968	4036	4047	4035	4041	4194	4135
autoconf-performance	680,3	1357,3	2020	2580,7	2953	3453,9	3737,7	3791,6	3697,1	3818,9	3768	3914	3855	3968	3895	4140	4138
autoconf-security	680,3	1366,	2013	2614,8	3008	3485,4	3637,0	3794,9	3807,6	3840,5	3798	3814	4061	4014	3879	4126	4167
blackhole	704,7	1421,5	2050	2714,3	3086	3444,6	3841,0	3887,5	3890,5	3932,3	3836	4018	3912	4129	4074	4200	4062
no-simult-connect	708,0	1418,8	2074	2669,0	3132	3561,1	3753,5	3836,1	3888,	3814,5	3972	3904	3884	4058	3973	4220	4283
noexec	682,9	1366,1	1997	2565,0	3014	3486,4	3774,2	3795,5	3733,7	3936,9	3817	4053	3894	4155	3930	4094	4042
noexec-pageexec	682,5	1377,0	2019	2608,0	3033	3376,0	3765,7	3745,3	3837,3	3713,2	3918	3831	3968	3953	3999	4042	4199
noexec-segmexec	682,9	1360,6	1964	2611,7	2969	3464,4	3791,1	3686,4	3843,8	3830,9	3790	3980	3842	4129	3928	4098	3977
rap	713,9	1429,5	2130	2731,1	3067	3567,7	3873,2	3910,8	3923,4	3849,7	3978	3970	4074	4099	4094	4168	4186
rbac	713,1	1429,3	2103	2715,4	3104	3548,2	3815,8	3892,9	3903,0	3862,0	3976	4002	4094	4095	4001	4265	4183
sanitize	699,3	1412,5	2074	2702,1	3172	3547,9	3733,2	3885,1	3890,2	3860,0	3836	3982	4045	3943	4131	4153	4108
socket-limit	706,8	1414,7	2063	2703,5	3191	3530,0	3836,6	3790,0	3958,7	3951,0	3987	4014	4039	4109	4047	4225	4099
stackleak	695,5	1401,7	2072	2639,6	3035	3474,0	3635,2	3806,1	3701,7	3823,8	3819	4139	3837	4073	3941	4002	4080
structleak	708,2	1419,8	2060	2656,9	3148	3433,	3843,8	3775,4	3911,8	3892,4	3858	4031	4023	4120	4161	4107	4130
uderef	650,9	1293,6	1927	2478,4	2811	3333,3	3606,2	3608,3	3655,3	3721,9	3705	3704	3744	3858	3839	3863	3943

Tabulka 13.99: Absolutní srovnání konfigurací

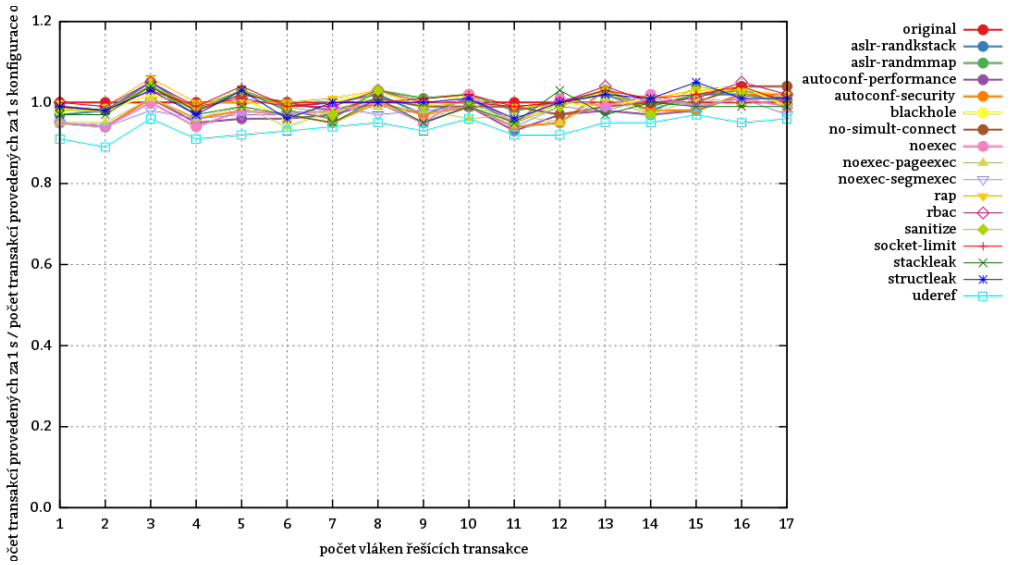


Obrázek 13.32: Absolutní srovnání konfigurací

13.8.9.2 Srovnání konfigurací

měřená konfigurace	počet vláken řešících transakce 1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	průměr
original	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
aslr-randkstack	0,97	0,98	1,05	0,98	1,03	0,99	1	1,02	0,97	1,01	0,95	1	1,02	0,98	1,02	1,02	1	0,99
aslr-randmmap	0,99	0,98	1,04	0,98	1,03	0,98	0,96	1,03	1,01	1,02	0,98	1	1,03	0,99	1,02	1,03	1,01	1
autoconf-performance	0,95	0,94	1,01	0,95	0,96	0,96	0,97	1	0,95	0,99	0,93	0,97	0,98	0,97	0,98	1,02	1,01	0,97
autoconf-security	0,95	0,94	1,01	0,96	0,98	0,97	0,95	1	0,97	0,99	0,94	0,95	1,03	0,98	0,98	1,02	1,02	0,97
blackhole	0,98	0,98	1,02	0,99	1,01	0,96	1	1,03	0,99	1,02	0,95	1	0,99	1,01	1,03	1,04	0,99	0,99
no-simult-connect	0,99	0,98	1,04	0,98	1,02	1	0,98	1,01	0,99	0,99	0,99	0,97	0,99	1	1	1,04	1,04	1
noexec	0,95	0,94	1	0,94	0,98	0,97	0,98	1	0,95	1,02	0,95	1,01	0,99	1,02	0,99	1,01	0,99	0,98
noexec-pageexec	0,95	0,95	1,01	0,96	0,99	0,94	0,98	0,99	0,98	0,96	0,97	0,95	1,01	0,97	1,01	1	1,02	0,97
noexec-segmexec	0,95	0,94	0,98	0,96	0,97	0,97	0,99	0,97	0,98	0,99	0,94	0,99	0,98	1,01	0,99	1,01	0,97	0,97
rap	1	0,99	1,06	1	1	1	1,01	1,03	1	1	0,99	0,99	1,03	1,01	1,03	1,03	1,02	1,01
rbac	1	0,99	1,05	0,99	1,01	0,99	0,99	1,03	1	1	0,99	1	1,04	1	1,01	1,05	1,02	1
sanitize	0,98	0,98	1,04	0,99	1,03	0,99	0,97	1,03	0,99	1	0,95	0,99	1,03	0,97	1,04	1,02	1	1
socket-limit	0,99	0,98	1,03	0,99	1,04	0,99	1	1	1,01	1,02	0,99	1	1,03	1,01	1,02	1,04	1	1
stackleak	0,97	0,97	1,04	0,97	0,99	0,97	0,95	1,01	0,95	0,99	0,95	1,03	0,97	1	0,99	0,99	0,99	0,98
structleak	0,99	0,98	1,03	0,97	1,03	0,96	1	1	1	1,01	0,96	1	1,02	1,01	1,05	1,01	1,01	1
uderef	0,91	0,89	0,96	0,91	0,92	0,93	0,94	0,95	0,93	0,96	0,92	0,92	0,95	0,95	0,97	0,95	0,96	0,93

Tabulka 13.100: Relativní srovnání konfigurací vůči referenční konfiguraci original



Obrázek 13.33: Relativní srovnání konfigurací vůči referenční konfiguraci original

13.8.9.3 Zhodnocení testu

Měření ukázalo, že úpravy prováděné Grsecem mají jen mírný dopad na výkonnost DBMS: Největší dopad na výkonnost potvrdila, tak jako i v předešlých testech, konfigurace udereF, při jejíž aplikaci klesá výkon systému k průměrně 93 % výkonu systému s referenční konfigurací. Prudký růst počtu provedených transakcí se zastavuje u 7 do vláken řešících transakce. Poté roste již jen mírně. Příčinou tohoto zlomu je počet jader, kterými testovaný stroj disponuje. Na rozdíl od testu uvedeném v kapitole 13.8.1 zde však počet jader procesoru nepředstavuje ostrý limit a počet provedených transakcí stále stoupá a to přibližně lineárně.

13.8.10 Test sysbench_db_readwrite

Test `sysbench_db_readwrite` zjišťuje počet transakcí, které je databáze schopna vykonat za 1 s. Na rozdíl od testu 13.8.9 je v tomto testu do databáze také zapisováno. Dotazy tak již nejsou výhradně jen `SELECT`, ale i transakce, čili `BEGIN .. END`.

NZPV (x)	počet vláken řešících transakce
jednotka NZPV	není (bezrozměrná veličina)
ZPV (y)	počet transakcí provedených za 1 s
jednotka ZPV	není (bezrozměrná veličina)
počet opakování měření	1
počítač spouštějící testovací příkaz	xenmv

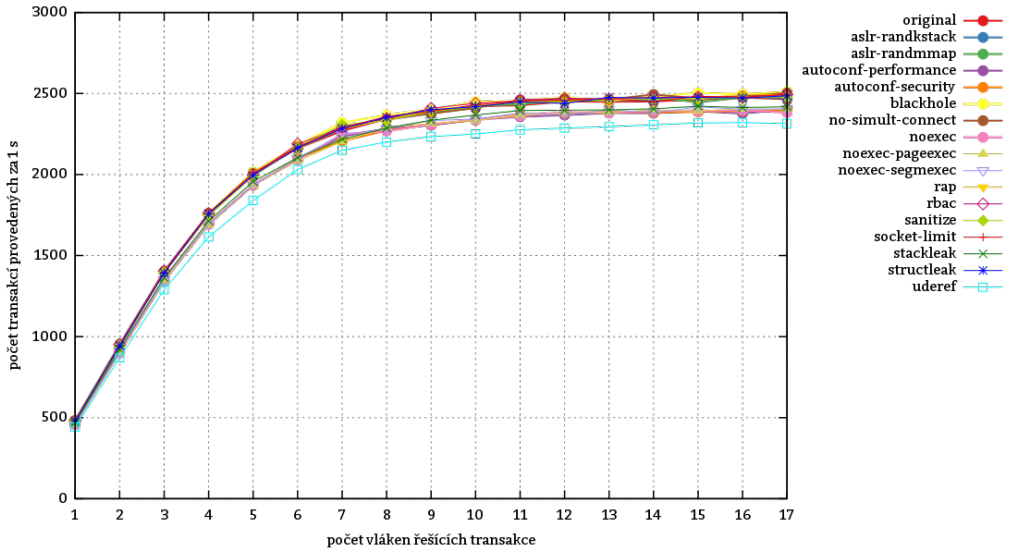
testovací příkaz	`sysbench --test=oltp --mysql-db=db --mysql-user=root`
	`↪ --mysql-password=+ --oltp-table-size=1000000 --max-requests=0`
	`↪ --db-driver=mysql --oltp-read-only=off --max-time=20`
	`↪ --num-threads=${X} run`

Tabulka 13.101: Charakteristiky testu sysbench_db_readwrite

13.8.10.1 Naměřené hodnoty pro jednotlivé konfigurace

měřená konfigurace	počet vláken řešících transakce																
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
original	480,36	938,6	1398,3	1750,3	2004	2161,6	2269,4	2343,7	2389,0	2421,3	2427,1	2450,7	2448,9	2451,6	2462,2	2474,7	2466,2
aslr-randkstack	477,	940,8	1399,7	1757,1	2006	2163,6	2283,1	2352,0	2377,6	2427,3	2436,4	2467,	2458,0	2460,7	2468,5	2478,3	2474,2
aslr-randmmap	482,6	954,6	1402,5	1758,1	2013	2171,4	2283,4	2343,8	2382,2	2421,9	2434,7	2454,3	2458,4	2474,5	2446,7	2474,3	2489,6
autoconf-performance	459,37	905,7	1343,7	1693,9	1935	2094,7	2234,6	2283,3	2304,1	2338,2	2356,1	2366,6	2380,4	2379,0	2390,6	2376,6	2400,7
autoconf-security	459,35	901,4	1350,0	1694,2	1934	2097,1	2207,7	2271,4	2309,8	2335,6	2369,1	2379,2	2387,8	2383,2	2387,1	2392,6	2392,5
blackhole	483,6	947,1	1400,5	1762,4	2013	2180,9	2321,1	2366,6	2400,6	2449,2	2452,7	2476,0	2456,0	2478,7	2504,0	2495,3	2510,5
no-simult-connect	480,12	942,6	1402,8	1760,1	2006	2170,8	2294,2	2339,4	2376,3	2410,6	2460,6	2466,6	2462,8	2493,7	2469,7	2483,3	2503,7
noexec	461,01	898,4	1344,1	1692,4	1939	2087,4	2250,3	2267,4	2309,4	2338,3	2361,3	2386,2	2380,2	2395,2	2391,4	2396,6	2381,8
noexec-pageexec	459,24	902,5	1344,6	1687,2	1933	2083,3	2204,2	2292,8	2315,9	2333,6	2365,0	2380,2	2384,1	2396,1	2393,3	2404,7	2402,5
noexec-segmexec	454,82	906,7	1350,5	1692,1	1932	2100,7	2246,0	2281,8	2333,5	2341,7	2378,4	2367,3	2400,2	2380,8	2417,3	2402,5	2406,0
rap	479,22	943,8	1402,1	1753,3	2009	2160,6	2283,6	2338,2	2397,5	2426,5	2444,6	2452,3	2472,3	2477,1	2469,4	2497,4	2491,6
rbac	482,2	951,8	1405,3	1760,2	1994	2189,1	2292,6	2343,9	2408,9	2439,7	2456,6	2470,1	2465,7	2474,8	2479,7	2477,5	2494,1
sanitize	477,15	939,4	1397,4	1746,4	1988	2167,5	2309,0	2332,8	2388,5	2421,0	2447,8	2449,9	2465,1	2465,8	2466,6	2479,7	2477,6
socket-limit	480,34	942,9	1396,6	1757,6	2009	2167,8	2288,8	2358,3	2389,9	2429,4	2454,5	2457,7	2468,0	2448,9	2482,9	2477,3	2494,3
stackleak	469,82	920,4	1363,6	1713,3	1956	2103,8	2218,1	2286,4	2335,3	2366,9	2395,0	2396,7	2398,9	2405,6	2421,3	2413,4	2418,0
structleak	480,58	943,1	1392,6	1757,4	1995	2163,7	2284,4	2352,1	2400,1	2418,7	2448,2	2438,6	2476,1	2472,1	2476,6	2472,2	2485,9
uderef	441,30	869,1	1289,6	1614,8	1839	2030,9	2149,8	2201,0	2234,3	2250,9	2275,7	2286,8	2295,0	2306,8	2317,8	2320,6	2314,7

Tabulka 13.102: Absolutní srovnání konfigurací

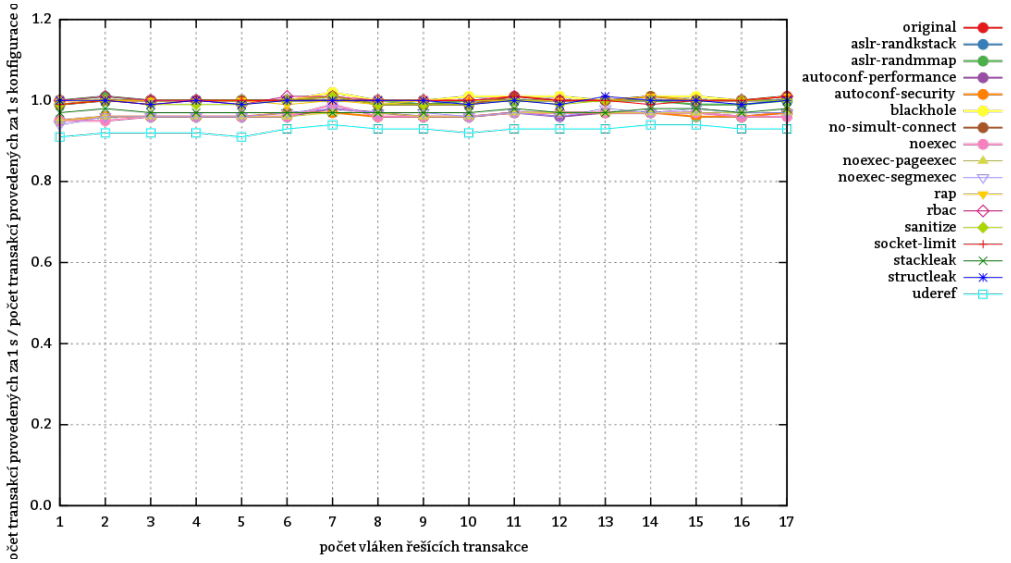


Obrázek 13.34: Absolutní srovnání konfigurací

13.8.10.2 Srovnání konfigurací

měřená konfigurace	počet vláken řešících transakce 1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	průměr
original	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
aslr-randkstack	0,99	1	1	1	1	1	1	1	0,99	1	1	1	1	1	1	1	1	0,99
aslr-randmmap	1	1,01	1	1	1	1	1	1	0,99	1	1	1	1	1	0,99	0,99	1	0,99
autoconf-performance	0,95	0,96	0,96	0,96	0,96	0,96	0,98	0,97	0,96	0,96	0,97	0,96	0,97	0,97	0,97	0,96	0,97	0,96
autoconf-security	0,95	0,96	0,96	0,96	0,96	0,97	0,97	0,96	0,96	0,96	0,97	0,97	0,97	0,97	0,96	0,96	0,97	0,96
blackhole	1	1	1	1	1	1	1,02	1	1	1,01	1,01	1,01	1	1,01	1,01	1	1,01	1
no-simult-connect	0,99	1	1	1	1	1	1,01	0,99	0,99	0,99	1,01	1	1	1,01	1	1	1,01	1
noexec	0,95	0,95	0,96	0,96	0,96	0,96	0,99	0,96	0,96	0,96	0,97	0,97	0,97	0,97	0,97	0,96	0,96	0,96
noexec-pageexec	0,95	0,96	0,96	0,96	0,96	0,96	0,97	0,97	0,96	0,96	0,97	0,97	0,97	0,97	0,97	0,97	0,97	0,96
noexec-segmexec	0,94	0,96	0,96	0,96	0,96	0,97	0,98	0,97	0,97	0,96	0,97	0,96	0,98	0,97	0,98	0,97	0,97	0,96
rap	0,99	1	1	1	1	0,99	1	0,99	1	1	1	1	1	1,01	1	1	1,01	0,99
rbac	1	1,01	1	1	0,99	1,01	1,01	1	1	1	1,01	1	1	1	1	1	1,01	1
sanitize	0,99	1	0,99	0,99	0,99	1	1,01	0,99	0,99	0,99	1	0,99	1	1	1	1	1	0,99
socket-limit	0,99	1	0,99	1	1	1	1	1	1	1	1,01	1	1	0,99	1	1	1,01	0,99
stackleak	0,97	0,98	0,97	0,97	0,97	0,97	0,97	0,97	0,97	0,97	0,98	0,97	0,97	0,98	0,98	0,97	0,98	0,97
structleak	1	1	0,99	1	0,99	1	1	1	1	0,99	1	0,99	1,01	1	1	0,99	1	0,99
uderef	0,91	0,92	0,92	0,92	0,91	0,93	0,94	0,93	0,93	0,92	0,93	0,93	0,93	0,94	0,94	0,93	0,93	0,92

Tabulka 13.103: Relativní srovnání konfigurací vůči referenční konfiguraci original



Obrázek 13.35: Relativní srovnání konfigurací vůči referenční konfiguraci original

13.8.10.3 Zhodnocení testu

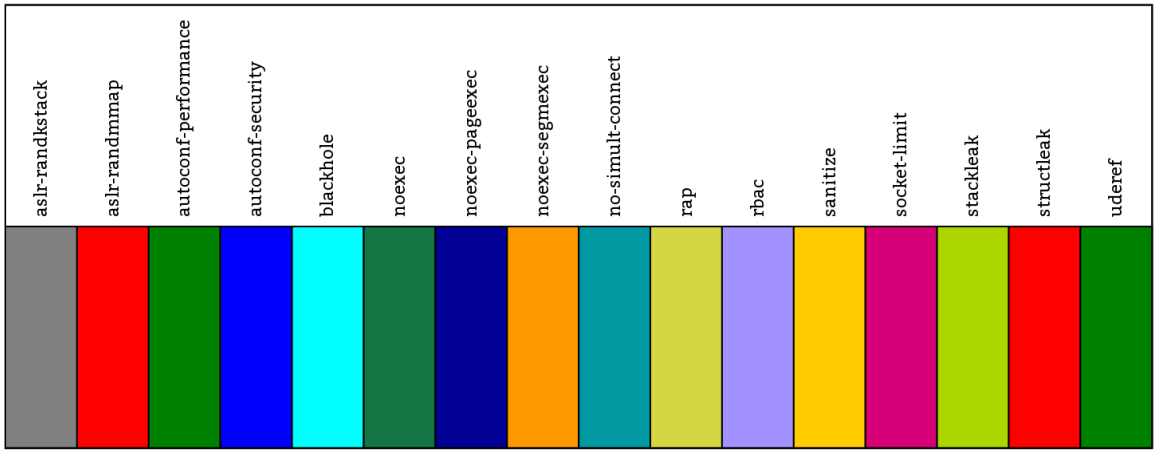
Test 13.8.10 ukázal podobný výsledek jako test 13.8.9. Z naměřených výsledků je dále patrné, že počet transakcí provedených za 1 s přestává škálovat. Počet provedených transakcí oproti měření v kapitole 13.8.9 v důsledku přítomnosti zápisu poklesl o cca 60 %.

14. Hodnocení

Prostupuje-li čtenář dokumentem sekvenčně, měl by být již seznámen s hrozbami i technikami proti nim stojícími v rámci Grsec. Předchozí kapitola uvedla, jak velký výkonnostní dopad mají jednotlivé techniky. Nyní nezbývá než vyslovit doporučení k jednotlivým z nich. Doporučení jednotlivých voleb následuje ale až po závěrečném srovnání, které pro čtenáře shrnuje naměřená data.

14.1 Srovnání bezpečnostních řešení

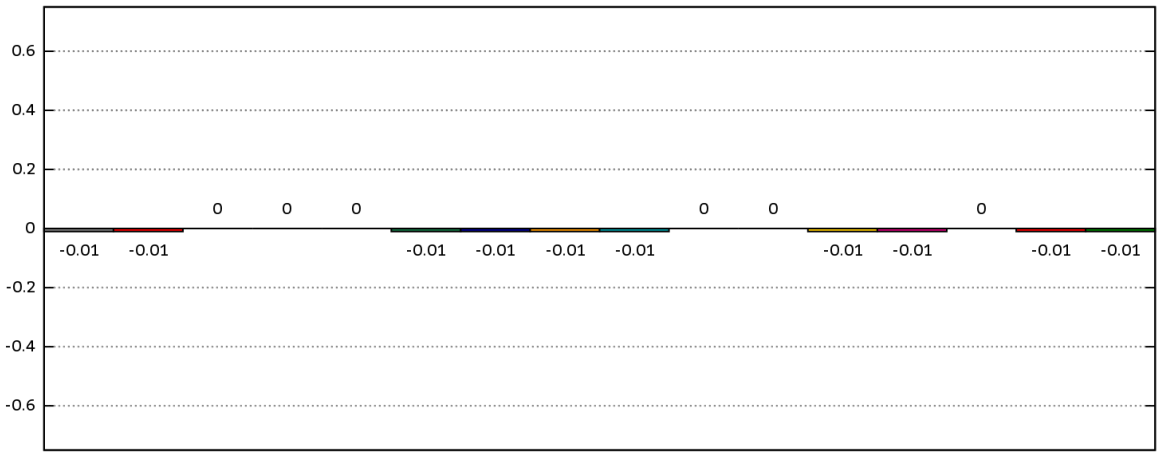
Na obrázcích uvedených v této kapitole je graficky znázorněn relativní rozdíl naměřené výkonnosti mezi jednotlivými konfiguracemi měřícího/měřeného prostředí. Jedná se o tatáž data, která již byla čtenáři předložena v kapitole 13 v tabulkách zachycujících relativní srovnání konfigurací. Každá konfigurace je reprezentována buďto průměrem z relativních hodnot jí přináležejících v příslušném testu či přímo konkrétní relativní hodnotou, bylo-li v daném testu prováděno pro každou konfiguraci měření pouze jedné hodnoty. Hodnotě 0 odpovídá výkonnost systému v konfiguraci original. Proto je daná konfigurace měřícího/měřeného systému ze srovnání odebrána.



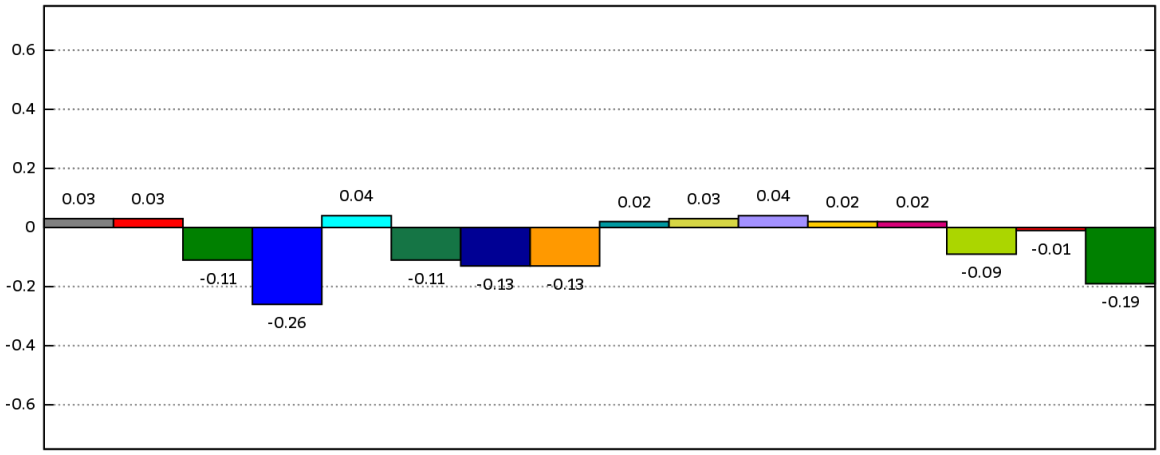
Obrázek 14.1: Klíč k určování testů ve srovnání

14.1.1 Srovnání jednotlivých konfigurací v testech zaměřených na výkonnost procesoru

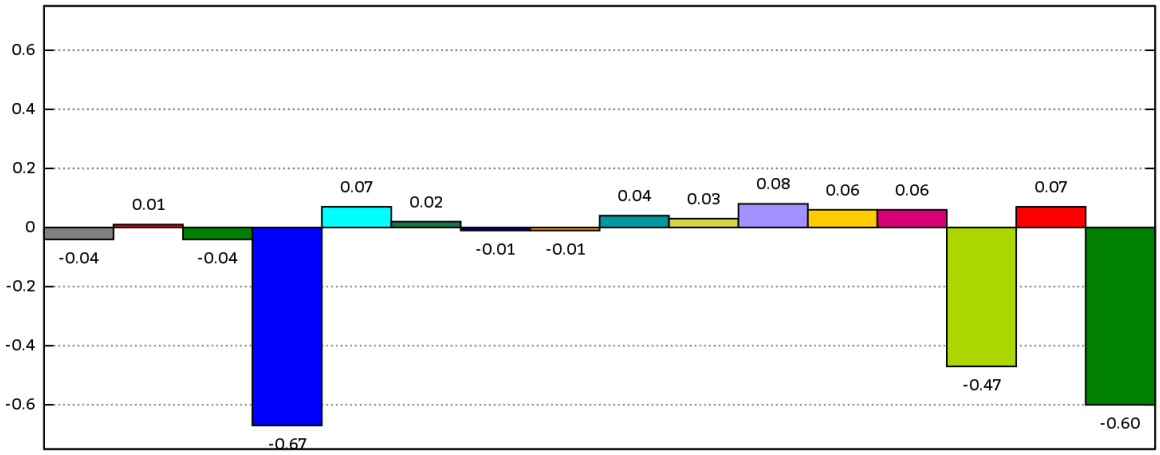
”



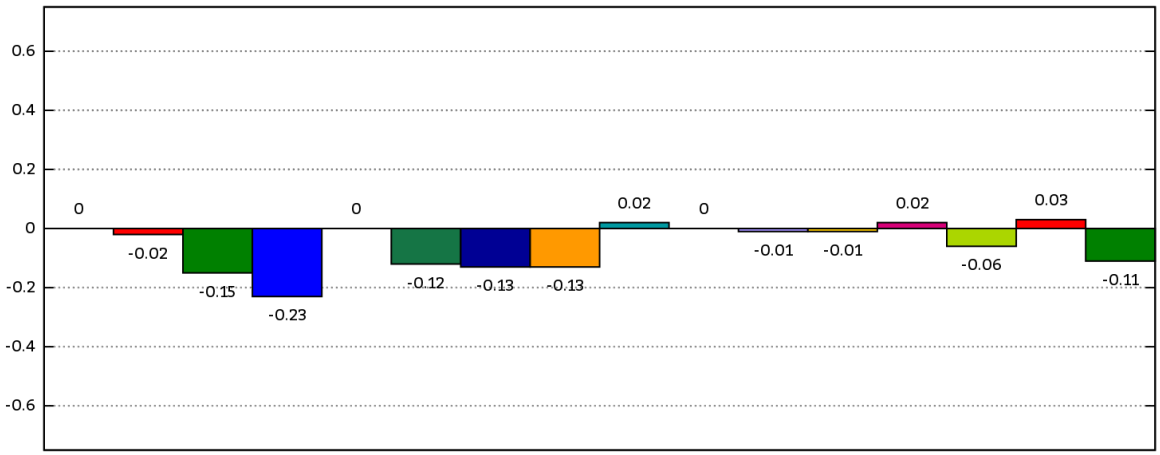
Obrázek 14.2: Srovnání konfigurací v testu `sysbench_cpu`



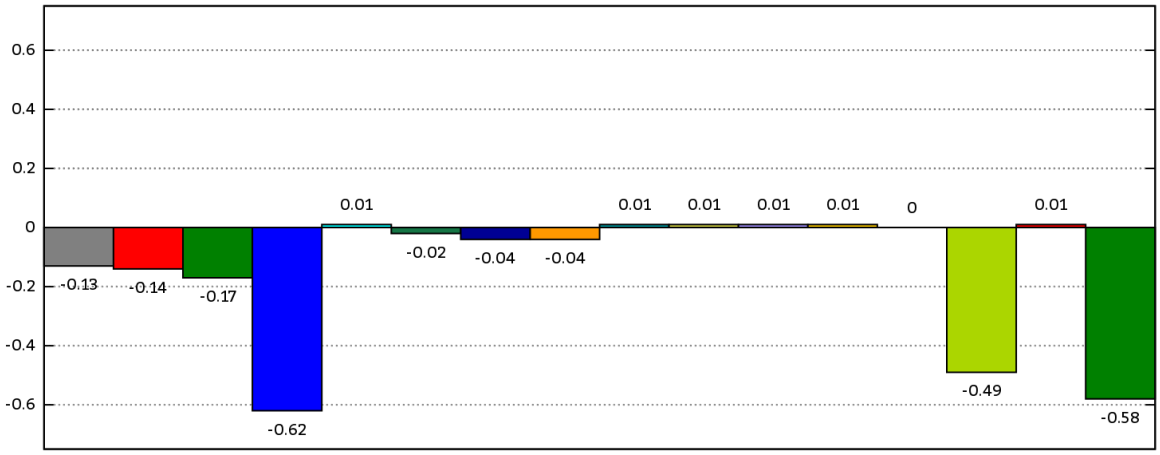
Obrázek 14.3: Srovnání konfigurací v testu `unixbench_context1`



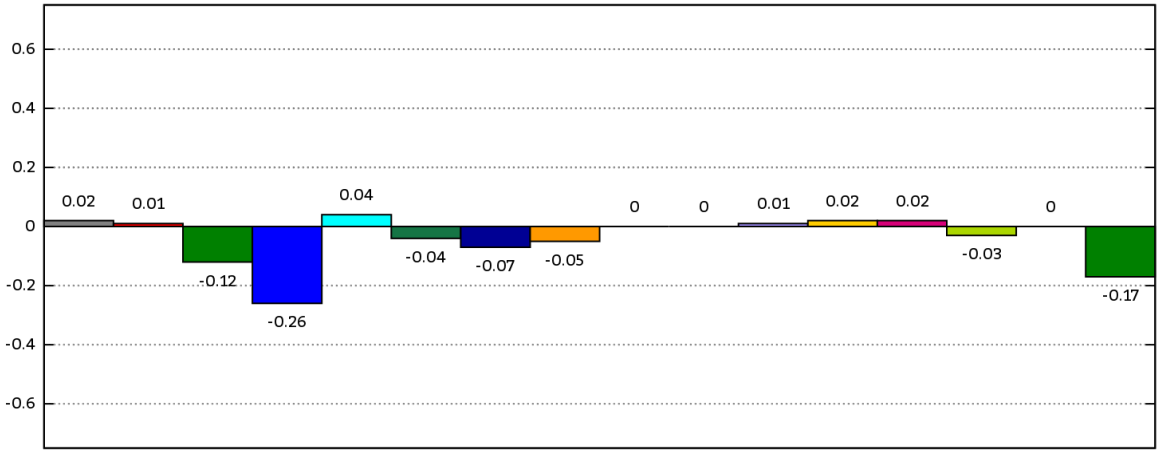
Obrázek 14.4: Srovnání konfigurací v testu `unixbench_pipe`



Obrázek 14.5: Srovnání konfigurací v testu `unixbench_spawn`



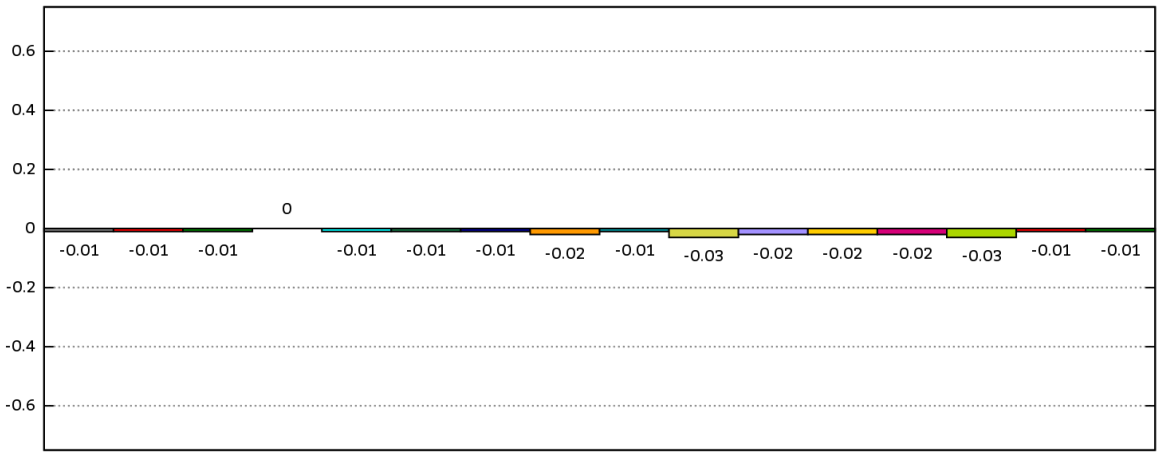
Obrázek 14.6: Srovnání konfigurací v testu `unixbench_syscall`



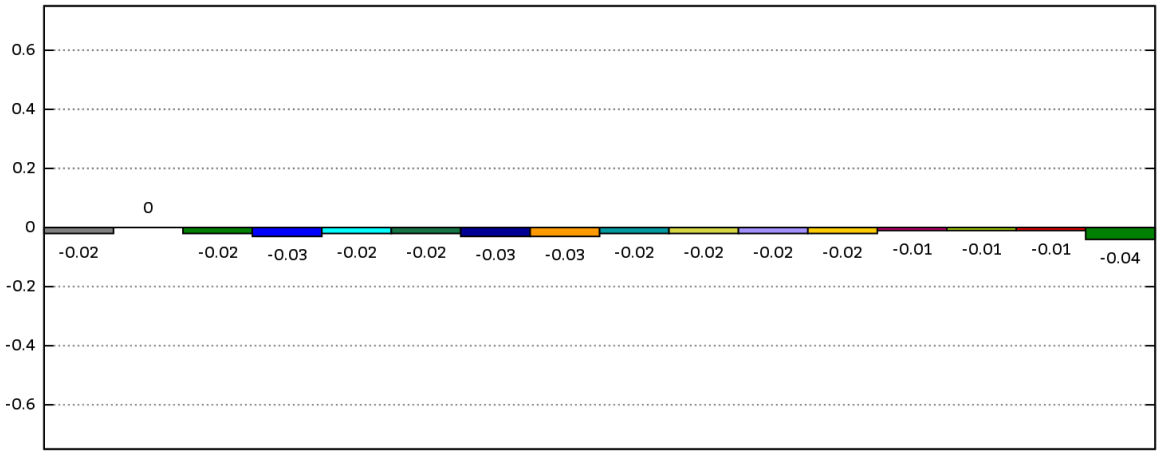
Obrázek 14.7: Srovnání konfigurací v testu `unixbench_execl`

14.1.2 Srovnání jednotlivých konfigurací v testech zaměřených na výkonnost hlavní paměti

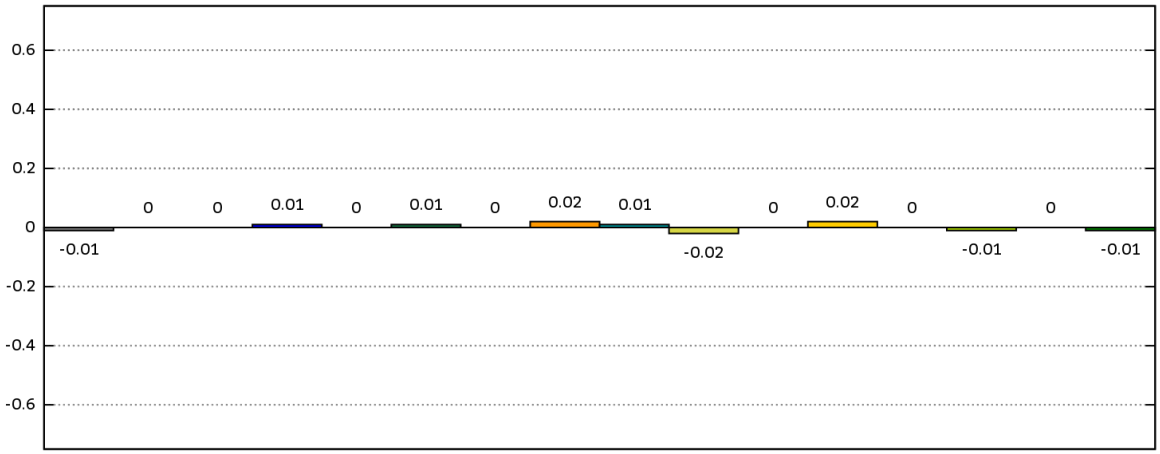
”



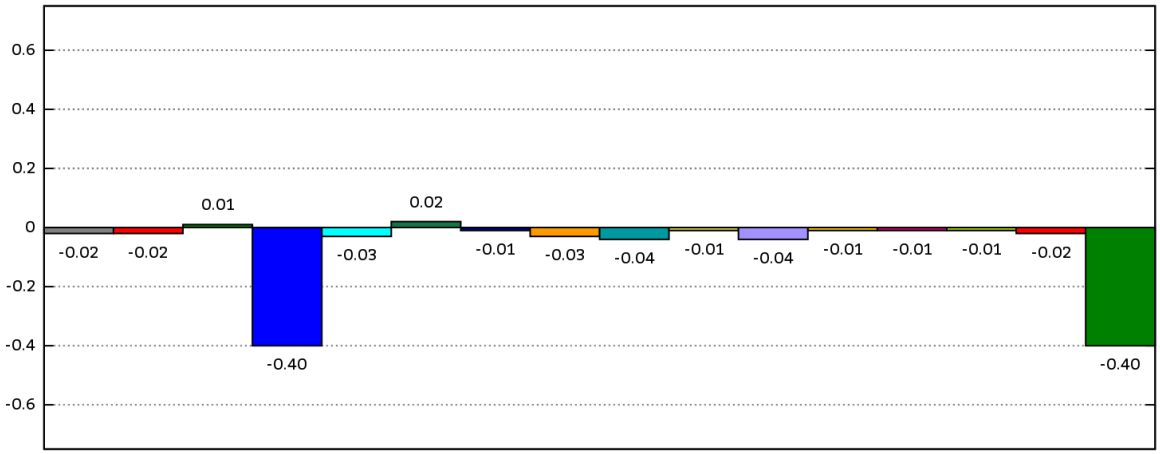
Obrázek 14.8: Srovnání konfigurací v testu `lmbench_bw_mem_rd`



Obrázek 14.9: Srovnání konfigurací v testu `sysbench_memory_write_seq`



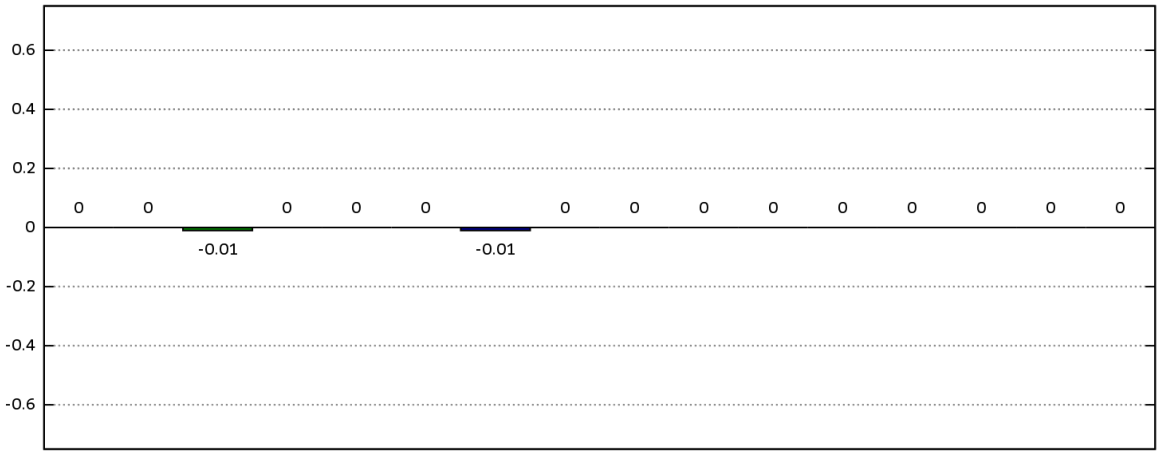
Obrázek 14.10: Srovnání konfigurací v testu `lmbench_bw_mem_wr`



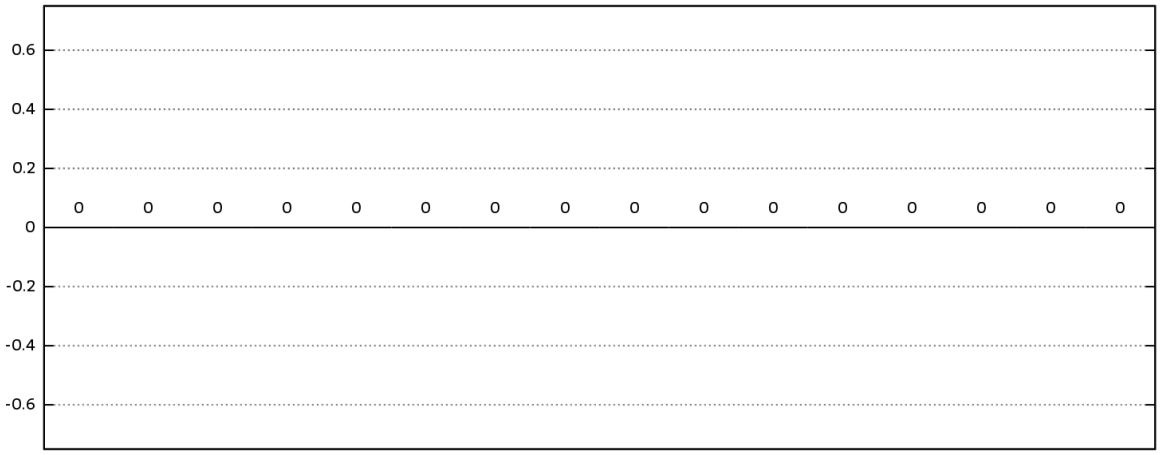
Obrázek 14.11: Srovnání konfigurací v testu `lmbench_bw_file_read_io_only`

14.1.3 Srovnání jednotlivých konfigurací v testech zaměřených na výkonnost sítě

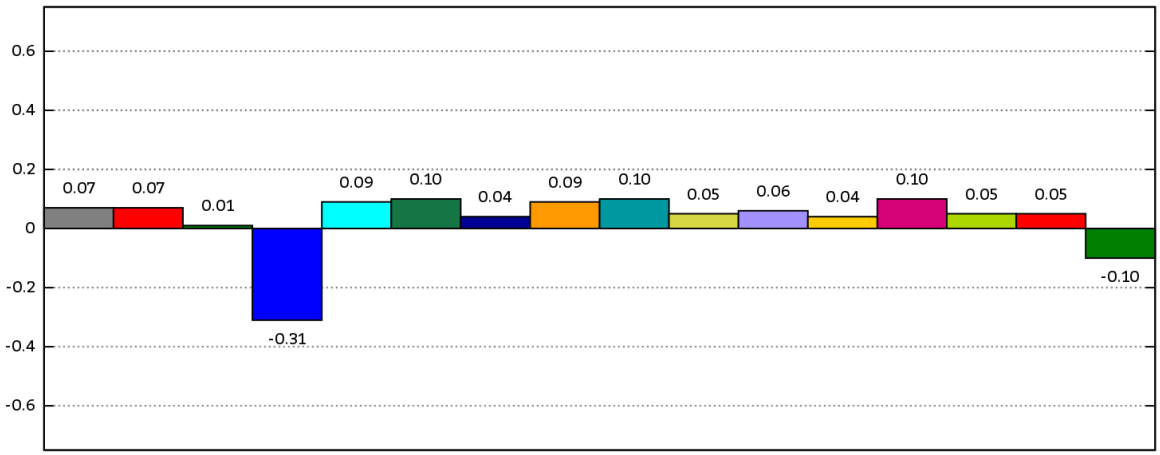
”



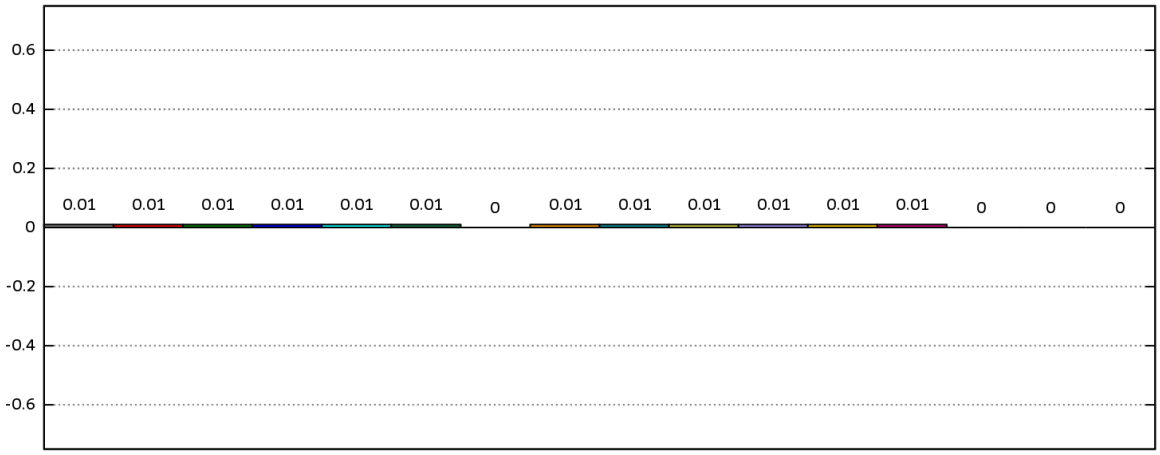
Obrázek 14.12: Srovnání konfigurací v testu `qperf_remote_bw_tcp`



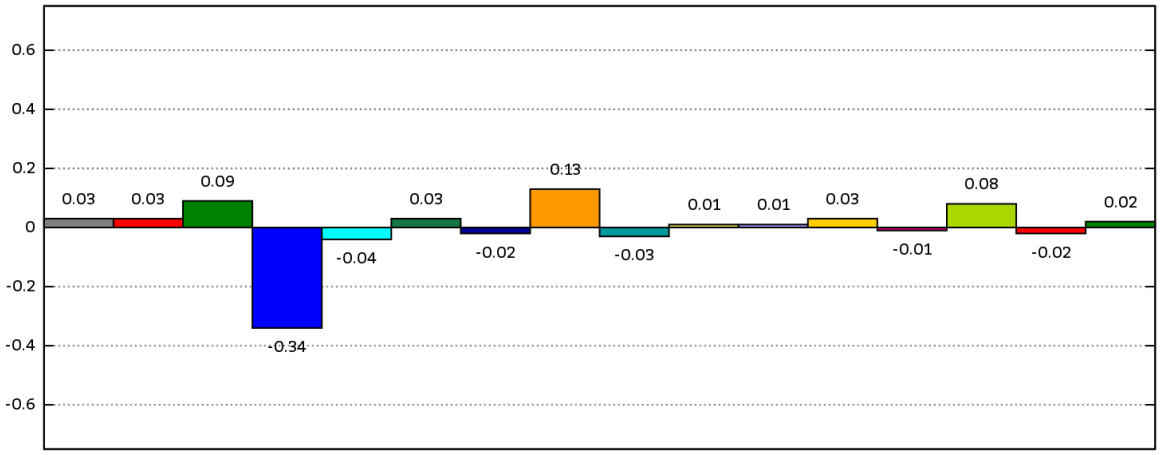
Obrázek 14.13: Srovnání konfigurací v testu `qperf_remote_bw_udp`



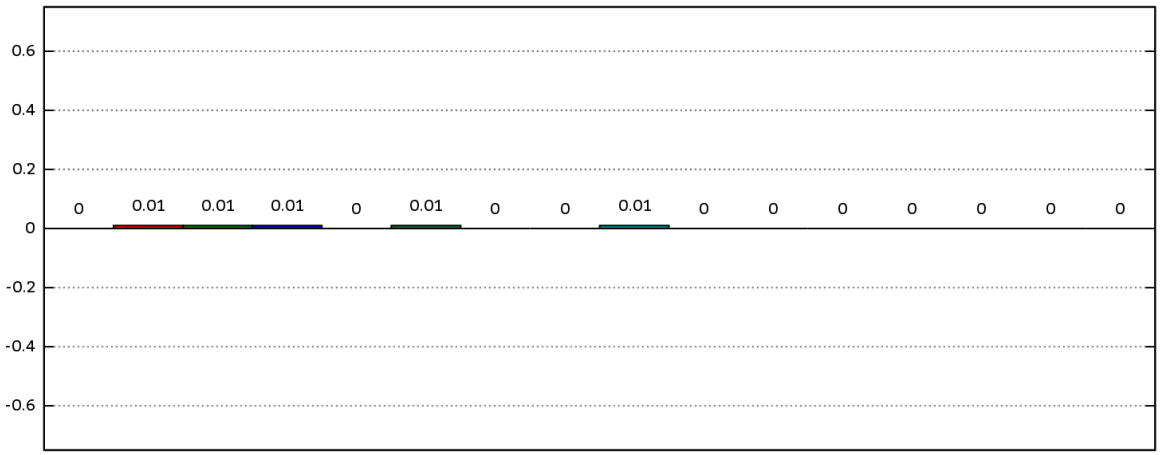
Obrázek 14.14: Srovnání konfigurací v testu `qperf_local_bw_tcp`



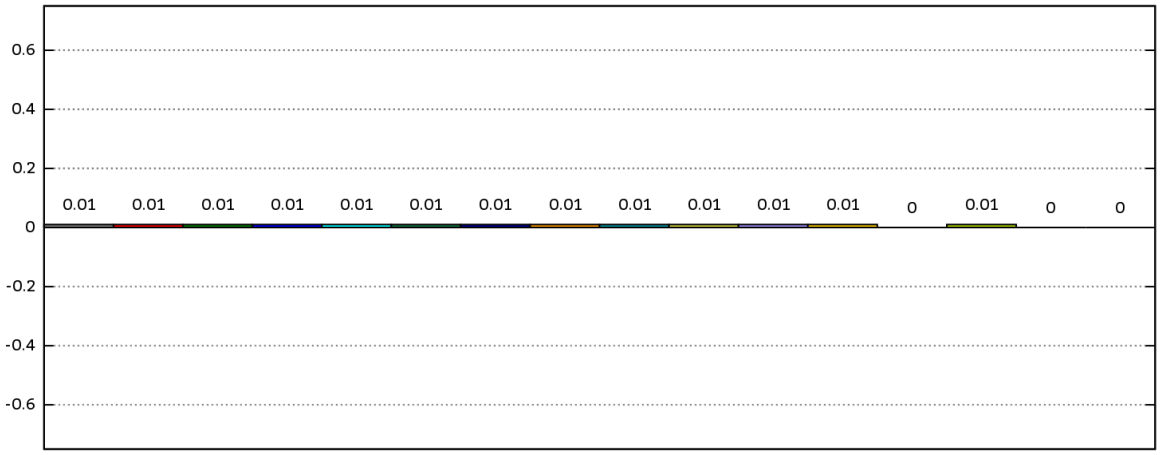
Obrázek 14.15: Srovnání konfigurací v testu `lmbench_lat_connect_remote`



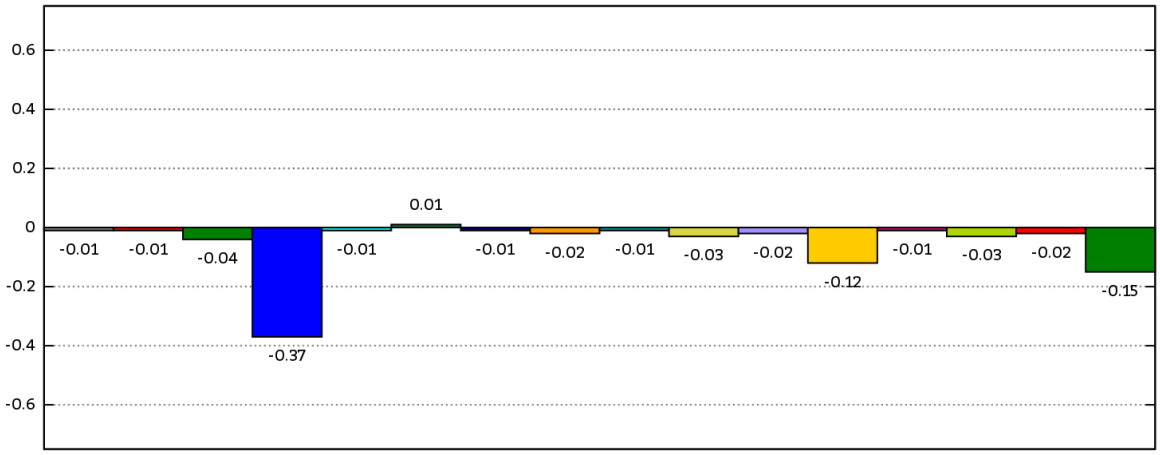
Obrázek 14.16: Srovnání konfigurací v testu `qperf_local_bw_udp`



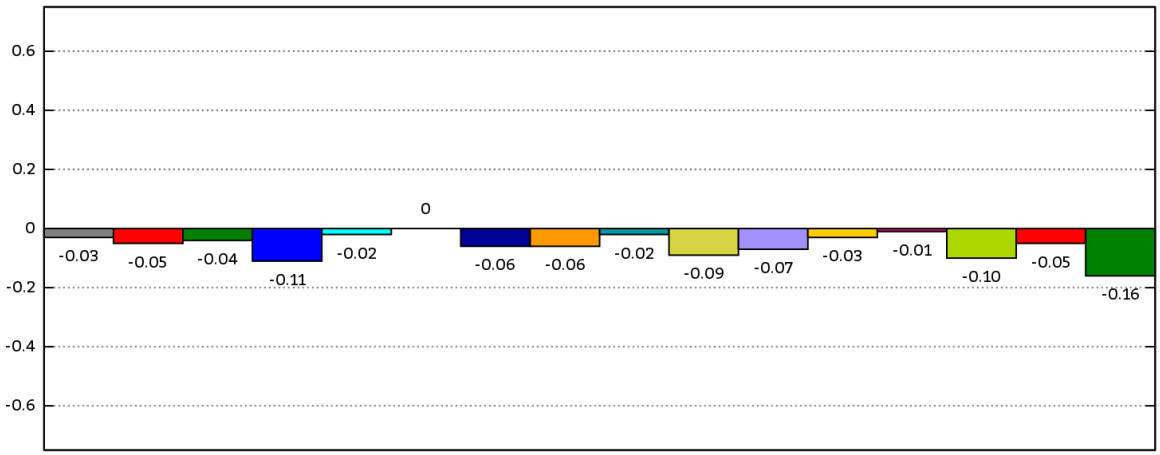
Obrázek 14.17: Srovnání konfigurací v testu `qperf_remote_lat_udp`



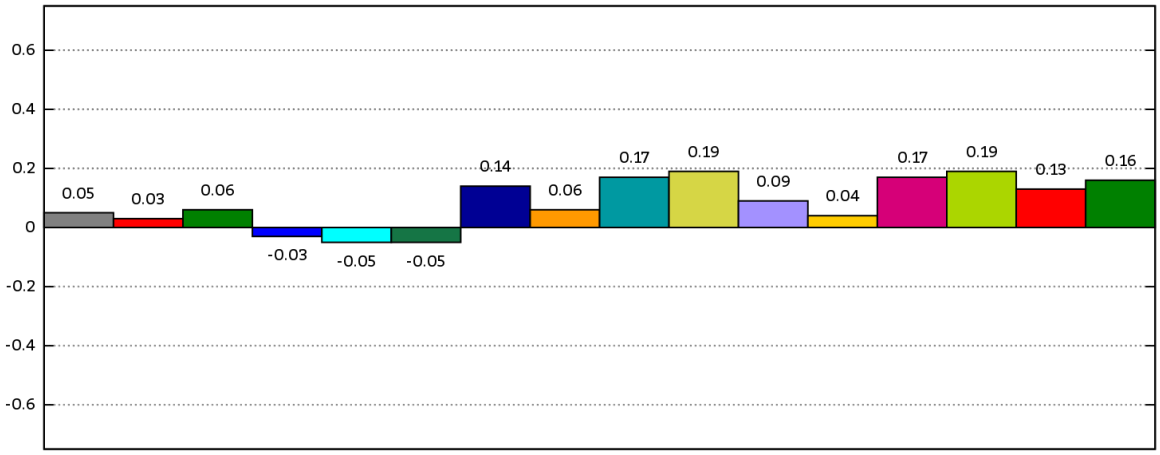
Obrázek 14.18: Srovnání konfigurací v testu `qperf_remote_lat_tcp`



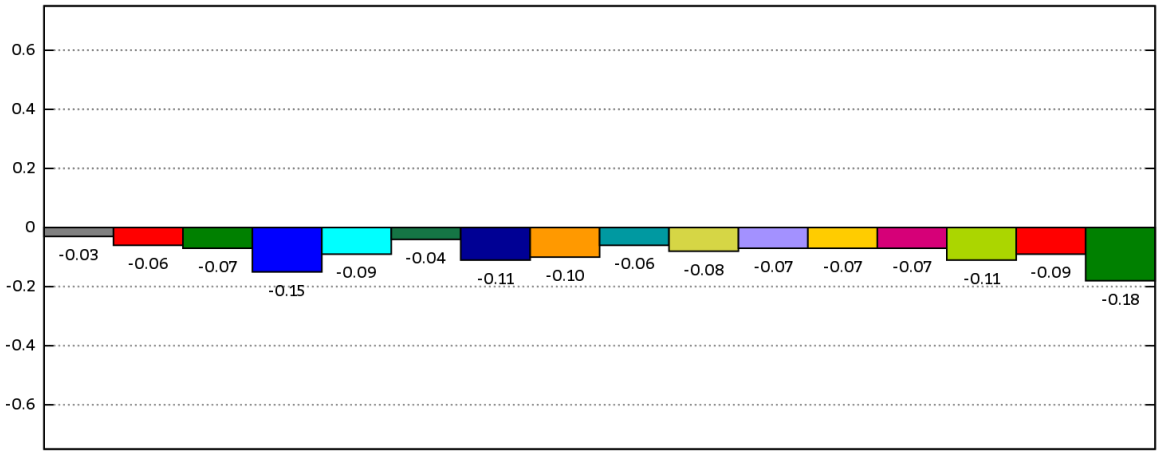
Obrázek 14.19: Srovnání konfigurací v testu `iperf_local_tcp`



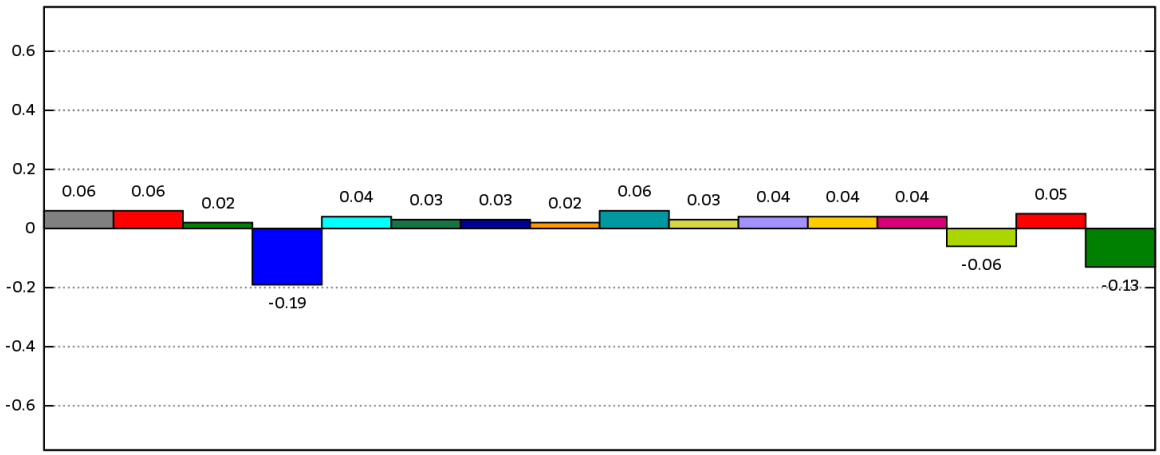
Obrázek 14.20: Srovnání konfigurací v testu `qperf_local_lat_tcp`



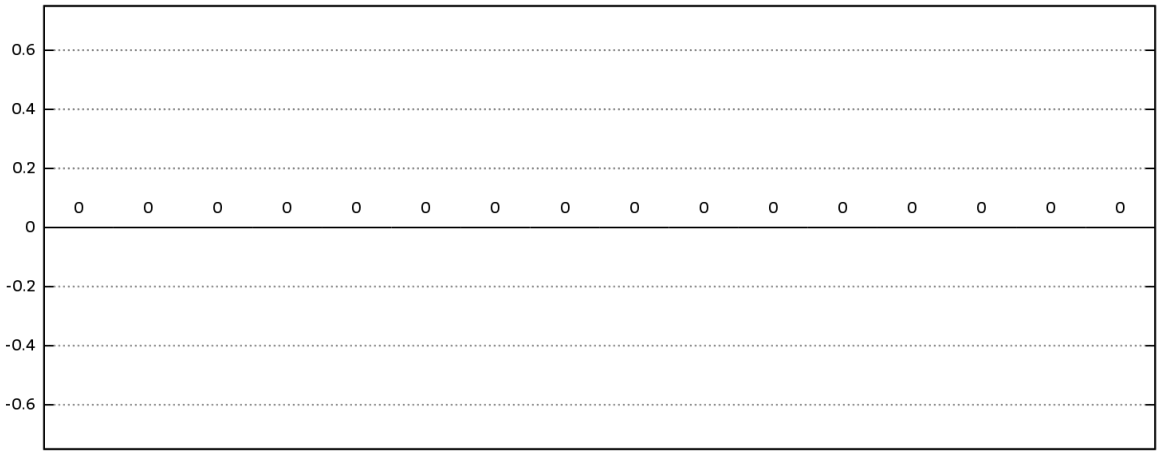
Obrázek 14.21: Srovnání konfigurací v testu `ping`



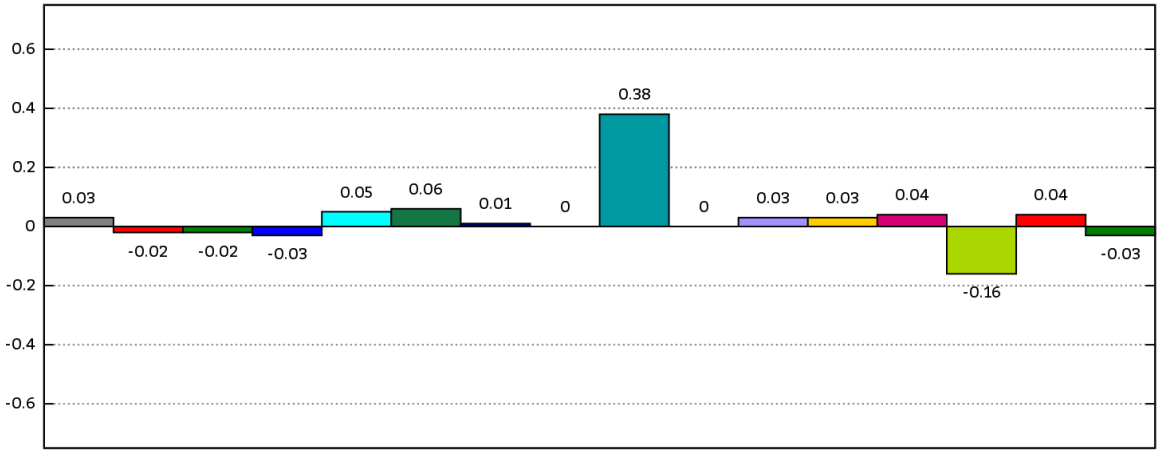
Obrázek 14.22: Srovnání konfigurací v testu `qperf_local_lat_udp`



Obrázek 14.23: Srovnání konfigurací v testu `lmbench_lat_unix_connect`



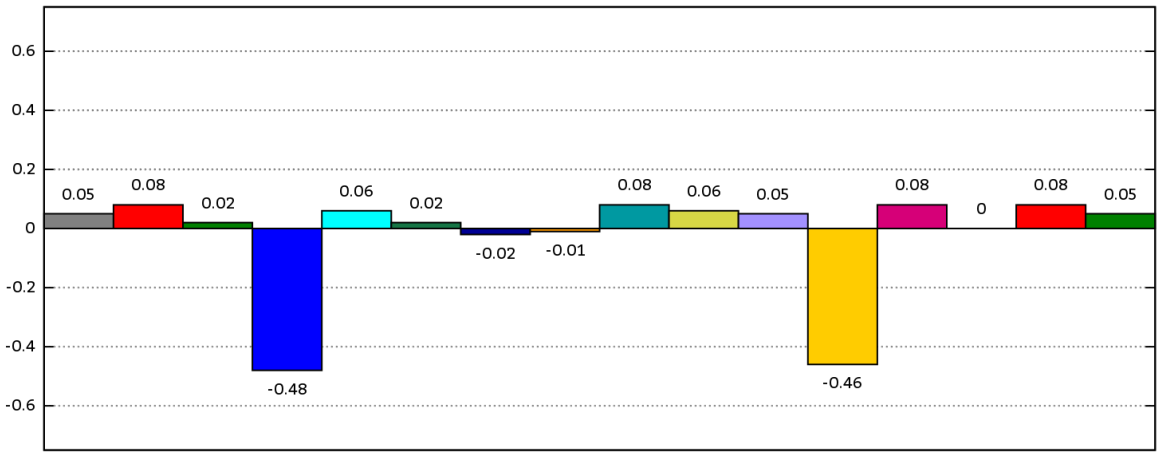
Obrázek 14.24: Srovnání konfigurací v testu `iperf_remote_tcp`



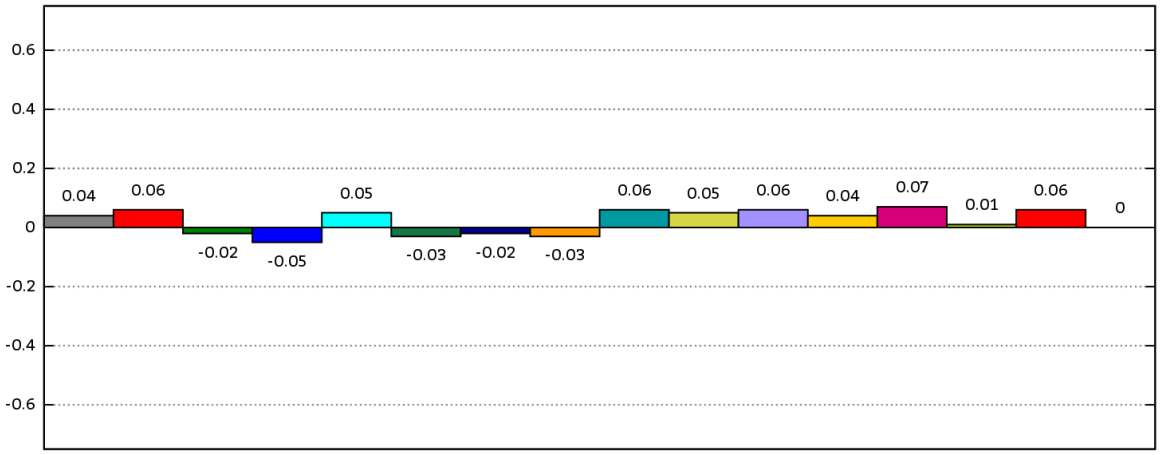
Obrázek 14.25: Srovnání konfigurací v testu `lmbench_lat_connect_local`

14.1.4 Srovnání jednotlivých konfigurací v testech zaměřených na výkonnost externí paměti

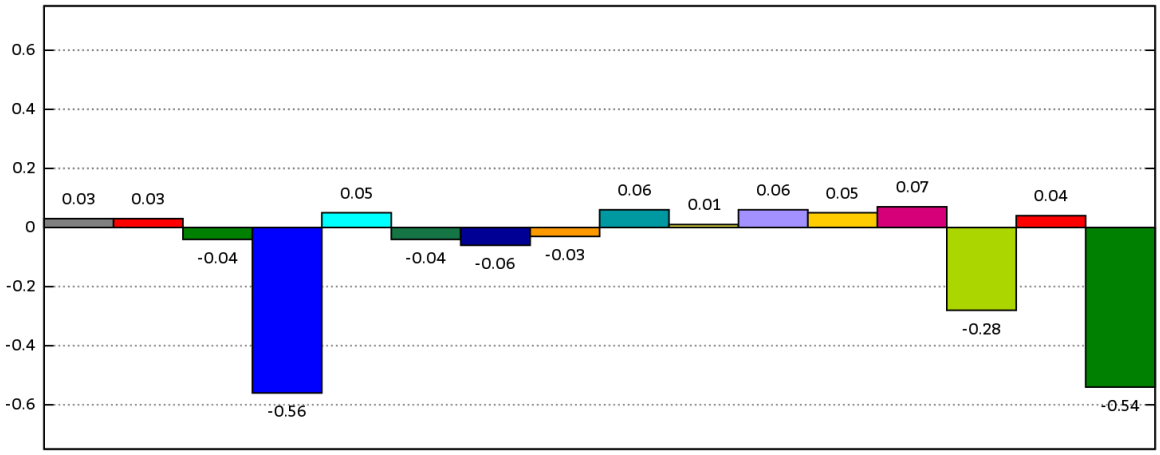
”



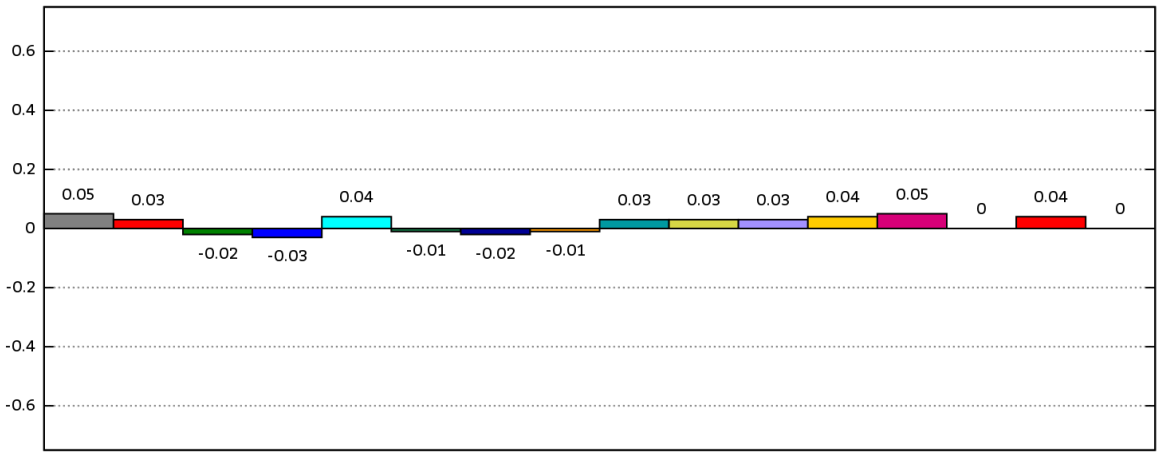
Obrázek 14.26: Srovnání konfigurací v testu `sysbench_fileio_rndwr_1`



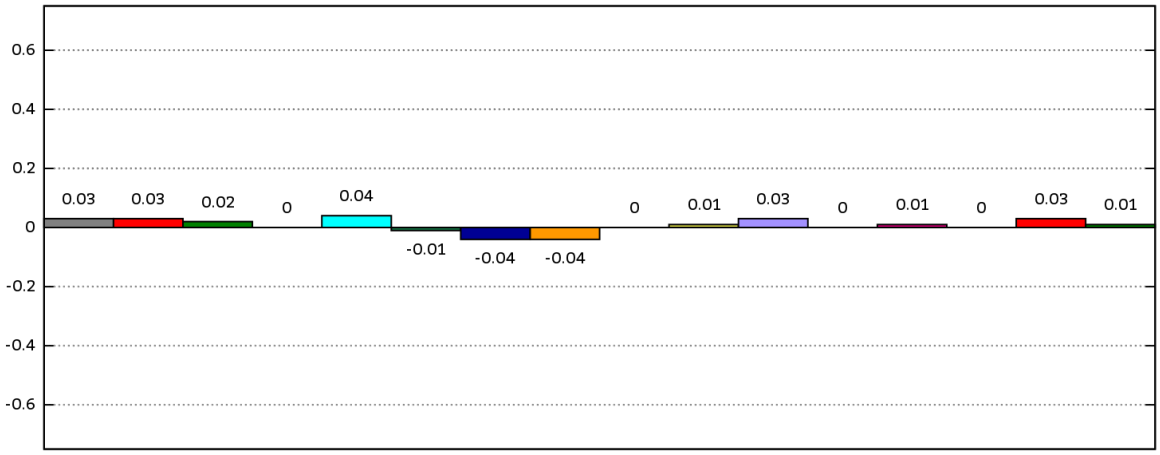
Obrázek 14.27: Srovnání konfigurací v testu `sysbench_fileio_rndwr_s`



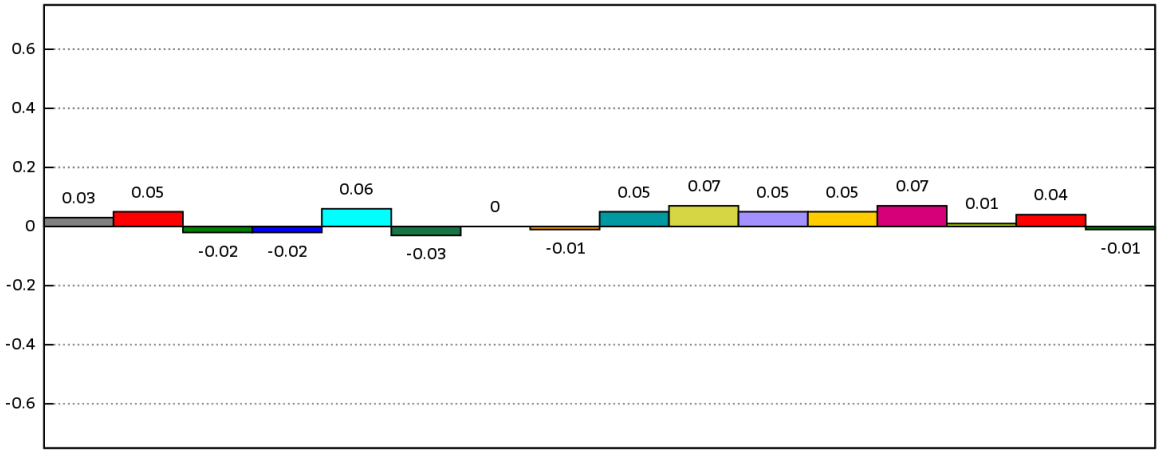
Obrázek 14.28: Srovnání konfigurací v testu `unixbench_fstime`



Obrázek 14.29: Srovnání konfigurací v testu `sysbench_fileio_rndrd_s`



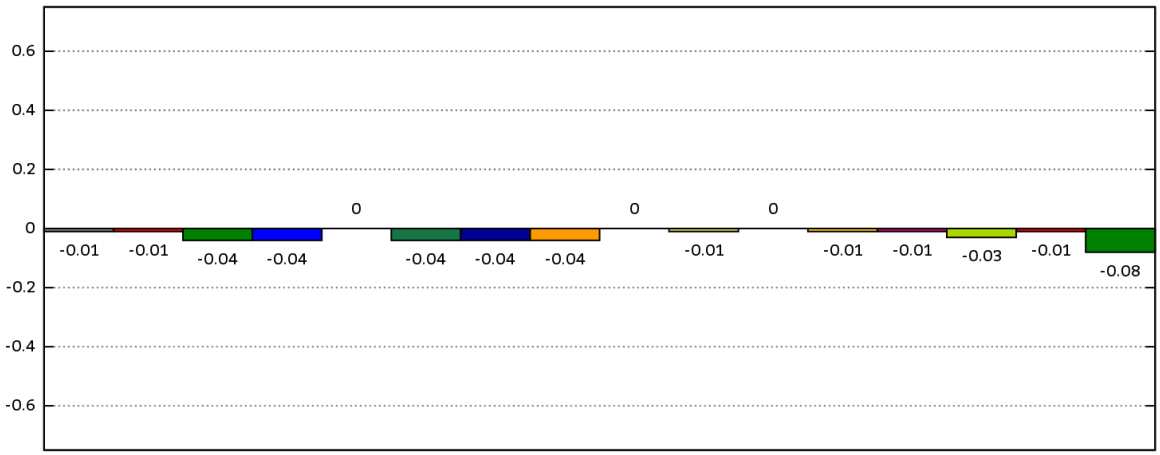
Obrázek 14.30: Srovnání konfigurací v testu `sysbench_fileio_seqwr_l`



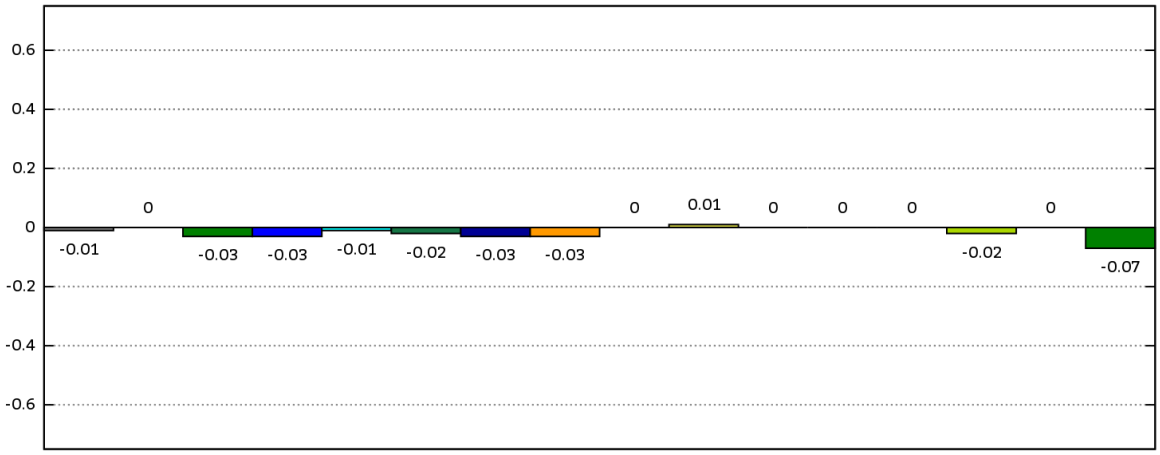
Obrázek 14.31: Srovnání konfigurací v testu `sysbench_fileio_seqrd_l`

14.1.5 Srovnání jednotlivých konfigurací v obecných testech

”



Obrázek 14.32: Srovnání konfigurací v testu `sysbench_db_readwrite`



Obrázek 14.33: Srovnání konfigurací v testu `sysbench_db_readonly`

14.2 Doporučení zabezpečení

Nastal čas, kdy z naměřených výsledků, které prezentovaly předešlé dvě kapitoly, a znalostí, které byly obsaženy v kapitolách jim předcházejících, je zapotřebí sestavit hodnocení či lépe odpověď na otázku: **Které volby zvolit?** Jelikož zabezpečení a výkonnost systému se vylučují, odpověď nemůže být jednoduchá. Proto je zapotřebí najít optimum. Těch však může být povícero, neboť vše se odvíjí od účelu zabezpečovaného systému. Zjednodušeně řečeno:

- **Volby, jejichž dopad na výkonnost systému je vždy pozitivní, zvolte.**

  Všechny volby přináší v nějakém směru vyšší bezpečnost systému, na kterém jsou aplikovány. Pokud s ní přinesou zároveň i vyšší výkonnost, tak sem s nimi. Typické to však není. V testech (viz kapitola 13) se jako nejméně restriktivní, co se týče výkonnosti, jeví konfigurace noexec čili volba `PAX_NOEXEC`. Tu rozhodně lze doporučit zapnout za všech okolností.
- **Volby, které nemají dopad na výkonnost systému a zároveň představují potencionální možnost rozšíření zabezpečení či možností správy v budoucnu, zvolte.**

  Volby, které v současnosti nepřináší žádný užitek, není třeba zavrhovat, pokud systém nijak funkčně ani výkonnostně neomezují. Budoucí vývoj v oblasti bezpečnosti může být jejich přítomnosti nakloněn.
- **Volby, jejichž dopad na výkonnost systému umí být i negativní, studujte.**

  Volby zpravidla přinášejí společně se zvýšením bezpečnosti též snížení výkonnosti. Otázkou je, zda-li jej lze akceptovat. Záleží na mnoha okolnostech:
  - Na míře nebezpečnosti volbou eliminovaných zranitelností.
  - Na hodnotě systémem manipulovaných dat.
  - Na prostředí, v němž je zabezpečovaný systém nasazen.
  - Na rezervě výkonnosti systému.

  Zkrátka záleží na ceně, kterou je provozovatel systému, ochoten zaplatit. Doporučení zapnutí/nezapnutí jednotlivých voleb se tak liší dle konkrétního nasazení. Rozhodnutí je tak značně subjektivní. Subjektivní pohled autora tohoto dokumentu je zachycen v tabulce 14.2. Tabulka 14.2 nese doporučení autora pro různá nasazení, jež jsou podrobněji popsána tabulkou 14.1.

Nasazení	Popis nasazení
souborový server	Server poskytující uživatelům přístup k souborům, které jsou na něm uloženy. Často též nabízí pokročilé možnosti zálohování a sdílení těchto souborů mezi uživateli. Na počítačích provozujících tento server není vhodné ponižovat jejich výkon ve vstupně-výstupních operacích. Naopak výkon hlavní paměti a procesoru si může dovolit v zájmu dosažení bezpečnosti poklesnout.
terminálový server	Poskytovatel vzdáleného prostředí způsobilého vykonávat programy čili příkazového interpretru neboli shellu. Časté spouštění úloh, na počítačích provozujících tyto servery, vyžaduje vysoký procesorový výkon a dostatek rychlé paměti. Naproti tomu diskové operace skýtají možnost k výkonnostním ústupkům ve prospěch bezpečnosti. Tu se rozhodně nevyplatí podceňovat, neb **uživateli** disponujícímu přístupem k shellu se **otevírá velmi široké spektrum zranitelností**. Ty mohou být schovány v uživateli přístupných programech či přímo v příkazech vestavěných v shellu. Nejenže je v tomto případě zranitelností potencionálně více, nežli u jiných typů serverů, ale zároveň **uživatel není od serveru odstíněn žádným protokolem**, který by mu případné exploitování zranitelnosti mohl alespoň znepříjemňovat. Proto jsou počítače provozující terminálové servery nejzranitelnějšími vůbec a **pokles výkonnosti způsobený aktivací ochran je** právě u počítačů tohoto typu nejvíce **ospravedlnitelný**.
webový server	Počítače provozující tento typ serveru jsou druhým nejohroženějším druhem z vyjmenovaných. To proto, že se na nich **provozují často nepříliš otestované, rychle psané a málo dokumentované programy**, které zranitelnosti zpravidla přitahují. Možné hrozby však nedosahují takových výšin jako v případě terminálového serveru, neb **exploitování je ztěžováno použitým protokolem** – útočník nemá přímý přístup k shellu.
databázový server	Počítač provozující databázový server typicky vykonává kód DBMS poskytujícího uživatelům rozhraní pro práci s daty, která jsou na něm též uložena. Provoz databáze a vykonávání komplexních dotazů vyžaduje zejména mnoho rychlé hlavní paměti a dostatek procesorového výkonu, stejně tak jako i rychlou externí paměť. Databázové servery bývají zpravidla více uzavřené a proto je jejich ochrana snazší. **Na ústupky ve prospěch zvýšení bezpečnosti v této oblasti nebývá zpravidla prostor.**
jmenný server	Zabezpečení počítačů provozujících jmenného servery by mělo mít co nejmenší negativní dopad na výkonnost sítě, zatímco výkonnost externí paměti a procesoru je celkem nepodstatná.

Tabulka 14.1: Charakteristiky jednotlivých nasazení

Tabulka 14.2: Doporučení volby pro jednotlivé volby v závislosti na zabezpečovaném prostředí

Volba	Nasazení					Komentář
	souborový server	terminálový server	webový server	databázový server	jmenný server	
GRKERNSEC CONFIG PRIORITY PERF	✓	✓	✓	✓	✓	Volba představuje zkratku pro volbu celé množiny voleb patche Grsec/PaX naráz. Tato množina voleb je sestavena samotnými tvůrci patche a je optimalizována pro zvýšení bezpečnosti v oblastech, které nevyžadují významné ústupky z výkonnosti systému. Měření neprokázala významný úbytek výkonnosti. Proto není důvod volbu nezapnout.
GRKERNSEC CONFIG PRIORITY SECURITY	✗	✗	✗	✗	✗	Volba představuje zkratku pro volbu celé množiny voleb patche Grsec/PaX naráz. Tato množina voleb je sestavena samotnými tvůrci patche a je optimalizována pro nekompromisní zvýšení bezpečnosti. Dopad na výkonnost zabezpečovaného systému tomu odpovídá. Aktivace této volby je na místě, je-li výkonnost zabezpečovaného systému dostatečně naddimenzována potřebám. Vzhledem k tomu, že volba GRKERNSEC_CONFIG_PRIORITY_SECURITY automaticky zapíná volby PAX_MEMORY_UDEREF a PAX_MEMORY_SANITIZE, které v testech prokázaly největší dopad na výkonnost systému ze všech, tak obecně zapnutí této volby nelze doporučit.
PAX NOEXEC	✓	✓	✓	✓	✓	Volba vynutí podporu pro nespustitelné stránky v jádře. Standardně je k ní hardware již uzpůsoben, nicméně jádro nespustitelné stránky využívá jen nedůsledně (viz kapitola 9.2.1.1). Důsledná ochrana proti spouštění obsahu nespustitelné paměti je jedním ze základních zabezpečení, neboť hrozeb zneužívajících tuto zranitelnost je přehršel. Drobný výkonnostní úbytek je v takovém případě nepodstatný. Více v kapitole 9.2.1.1. Zpomalení zjištěné testem v kapitole 13.7.3 měřícím rychlost přepínání kontextu procesů zdaleka významné natolik, aby to vedlo k nedoporučení volby.
PAX PAGEEXEC[1]	✓	✓	✓	✓	✓	Volba PAGEEXEC byla testována jen pro zjištění, jak moc velký dopad na výkonnost má emulace NX bitu na 32 bitovém systému. Více v kapitole 9.2.2.1. Nebyl zjištěn téměř žádný rozdíl oproti volbě NOEXEC. Výkonnostní dopad na systém ve všech měřených disciplínách je opět velmi malý a jediné výraznější zpomalení bylo zaznamenáno testem 13.7.3 měřícím rychlost přepínání kontextu procesů.

↓ Pokračování tabulky na další straně.

[2] Tuto volbu lze uplatnit pouze v 32 bitovém režimu.

Pokračování tabulky z předchozí strany. ↑

Volba	Nasazení					Komentář
	souborový server	terminálový server	webový server	databázový server	jmenný server	
PAX SEGMEXEC[3]	✓	✓	✓	✓	✓	Volba SEGMEXEC byla testována jen pro zjištění, jak moc velký dopad na výkonnost má emulace NX bitu na 32 bitovém systému. Více v kapitole 9.2.2.2. V měření si počínala stejně jako PAX_PAGEEXEC.
PAX RANDKSTACK	✓	✓	✓	✓	✓	Výkonnostní dopad znáhodňování polohy počátku kernelspace-stack při vytváření jemu odpovídajícího procesu/vlákna byl shledán nepatrným. Více v kapitole 9.1.8.
PAX RANDUSTACK	✓	✓	✓	✓	✓	Výkonnostní dopad znáhodňování polohy počátku userspace-stack při vytváření jemu odpovídajícího procesu/vlákna byl shledán nepatrným. Více v kapitole 9.1.7.
PAX RANDMMAP	✓	✓	✓	✓	✓	Znáhodňování umístění počátku – RANDMMAP –, dle kterého jsou mapovány úseky do paměti voláním mmap() nemá na výkonnost systému negativní dopad. Více v kapitole 9.1.5.
GRKERNSEC NO RBAC	✗	✗	✗	✗	✗	Sama o sobě tato volba nemá vliv na bezpečnost. Pouze odebírá ze systému možnost definovat MAC přístupová práva. Ty lze využít například při specifikaci co přesně konkrétní subjekt – program – může a co již nikoliv. Přítomnost/nepřítomnost podpory Grsecurity ACL nemá vliv na výkonnost a proto je vypnutí dané volby na místě. Více v kapitole 6.1.
GRKERNSEC BLACKHOLE	✓	✓	✓	✓	✓	Aktivace této volby způsobí, že systém nebude odpovídat na komunikaci přicházející na porty, na nichž žádný proces nenaslouchá, čímž omezí přísun informací potencionálním útočníkům na systém. Dopad na výkonnost nulový. Dopad na bezpečnost značný. Zapnout. Více v kapitole 9.19.
GRKERNSEC NO SIMULT CONNECT	✓	✓	✓	✓	✓	Volba, která jde přímo proti části TCP [RFC793] tím, že ze síťového zásobníku vypouští podporu pro souběžné spojení. Má velmi mírný vliv na výkonnost. Zranitelnost, kterou představuje souběžné spojení je natolik závažná, že zapnutí je na místě. Více v kapitole 9.19. Testy neprokázaly podstatné zpomalení systému dané touto volbou. V testu uvedeném v kapitole 13.6.1 aktivace této volby naopak znamenala podstatné snížení času potřebného pro navázání TCP spojení.

↓ Pokračování tabulky na další straně.

[4] Tuto volbu lze uplatnit pouze v 32 bitovém režimu.

Pokračování tabulky z předchozí strany. ↑

Volba	Nasazení					Komentář
	souborový server	terminálový server	webový server	databázový server	jmenný server	
`GRKERNSEC SOCKET ALL`	✓	✓	✓	✓	✓	Nemá vliv na výkon. Na bezpečnost přímo sice taky ne, avšak možnost deaktivovat přístup programu k síti pouhým jeho přiřazením do skupiny se může hodit. Více v kapitole 9.19.
`GRKERNSEC DMESG`	✓	✓	✓	✓	✓	Nezpůsobuje žádný výkonnostní pokles. Znedostupnění většiny logu jádra pro neprivilegované uživatele ztíží potenciálnímu útočníkovi získávání informací o napadaném systému. Zapnutí této volby lze jedině doporučit. Více v kapitole 9.20. Podobnou funkcionalitu poskytuje i systémová proměnná `kernel.dmesg_restrict` zavedená roku 2010 do jádra (od verze jádra 2.6.37) [Ros10]. Tato volba znepřístupní `/dev/kmsg` neprivilegovaným uživatelům celý za základě požadavku na capabilitu `CAP_SYS_ADMIN`.
`PAX MEMORY SANITIZE`	✗	✓	✓	✗	✓	Volba zajistí fyzické mazání paměťových stránek v okamžiku jejich uvolnění. Vzhledem k velkému dopadu na výkonnost diskových operací (viz kapitola 13.8.4) a náročnosti útoku, který vyžaduje přidělení jedné konkrétní paměťové stránky, jejíž obsah je pro útočníka zajímavý, je zapnutí této volby velmi na zvážení. Laciněji podobného výsledku dosáhne provoz aplikací, které kritický obsah v paměti fyzicky přepisují před jeho dealokací samy či se paměťových stránek zbavují jen velmi zřídkakdy. Pro vysoký výkonnostní úbytek diskových operací není tuto volbu doporučeno zapínat plošně. Více v kapitole 9.13. Volba se negativně projevila na propustnosti hlavní paměti (test 13.4.1) i propustnosti externí paměti (test 13.8.4).
`PAX MEMORY STACKLEAK`	✗	✓	✓	✗	✗	Volba maže obsah kernelspace-stack aby tak zabránila úniku informací z něj do User Space (viz kapitola 8.6). Volba má výrazně negativní dopad při častých systémových voláních, neboť návrat z každého systémového volání významně zpomaluje. Více v kapitole 9.14. V testech se volba `PAX_MEMORY_STACKLEAK` projevila, a to významně: • zhoršuje odezvu síťové komunikace (testy 13.5.3, 13.6.3), • má negativní vliv na zápis do paměti (test 13.6.6), • zpomaluje obsluhu systémových volání (test 13.7.1 a 13.7.5), • a v ojedinělých případech zpomaluje i diskové operace (test 13.7.6).

↓ Pokračování tabulky na další straně.

Pokračování tabulky z předchozí strany. ↑

Volba	Nasazení					Komentář
	souborový server	terminálový server	webový server	databázový server	jmenný server	
PAX MEMORY STRUCTLEAK	✓	✓	✓	✓	✓	Preventivní inicializace lokálních proměnných jádra vynucená volbou PAX_MEMORY_STRUCTLEAK způsobuje zanedbatelný výkonnostní propad. Proto je lze doporučit k aktivaci. Volba chrání proti zranitelnosti CVE-2013-2141. Více v kapitole 9.15.
PAX MEMORY UDEREF	✗	✓	✗	✗	✗	Volba má dopad na přesuny dat mezi Kernel Space a User Space. Její aktivace chrání systém před útoky ret2usr popsanými v kapitole 8.9.2. Nicméně, jak je již zmíněno v kapitole 8.9.2.1, ochrana aktivovaná touto volbou nezabrání vylepšené verzi útoků – Return to direct mapped memory. Více v kapitole 9.5.2. Výkonnostní pokles způsobený touto volbou je velmi výrazný napříč všemi testovanými oblastmi: • **propustnost hlavní paměti** testy 13.4.1, 13.5.1 a 13.6.6 • **zpoždění paketů síťové komunikace** testy 13.6.3, 13.5.3 a 13.5.4 • **propustnosti externí paměti** test 13.6.4 • **rychlosti obsluhy systémových volání** testy 13.7.1 až 13.7.6 Závěr je zřejmě zřejmý: Volba PAX_MEMORY_UDEREF je tím méně vhodná, čím více aplikace interaguje se svým okolím. Pokud aplikaci k životu stačí čistá práce v podobě numerických výpočtů ve svém vlastním User Space s minimem interakce s okolím, tak by případně aktivaci této volby šlo akceptovat.
PAX RAP	✓	✓	✓	✓	✓	Útok ROP (viz kapitola 8.8) je vylepšením „shellcode injection“, proti němuž je systém bráněn volbou NOEXEC. ROP představuje natolik závažnou hrozbu, že úbytek výkonnosti zapříčiněný zapnutím RAP se jednoznačně vyplatí obětovat.

Konec tabulky.

14.3 Posun systému v klasifikaci TCSEC způsobený patchem Grsec/PaX

V kapitole 2 na samém začátku tohoto dokumentu byl čtenář seznámen s existencí kritérií pro hodnocení bezpečnosti a s ní spjatým dokumentem TCSEC. Ten definoval třídy IS (od nejméně bezpečných): D, C1, C2, B1, B2, B3 a A1. GNU Linux bez dodatečných úprav spadá do kategorie C1 (viz kapitola 2.2.1.2). Lze se aplikací voleb Grsec dobrat posunu GNU Linux v hodnocení definovaném TCSEC?

Třída C2 požaduje po IS

vše, co požadovala třída C1,
: GNU Linux splňuje – viz [Pat].

jemněji konfigurovatelné nepovinné řízení přístupu (DAC),
: GNU Linux splňuje – viz kapitola 5.3.2.

rušení obsahu systémových prostředků při jejich znovupoužití,
: GNU Linux bez dodatečných úprav nesplňuje. S patchem Grsec s aktivovanou volbou PAX_MEMORY_SANITIZE však ano.

jednoznačnou identifikaci a autorizaci každého jednotlivého uživatele a
GNU Linux splňuje – viz kapitola 3.1.
logování z hlediska bezpečnosti významných událostí.
GNU Linux bez dodatečných úprav nesplňuje. S patchem Grsec s aktivovanými volbami `GRKERNSEC_EXECLOG`, `GRKERNSEC_RESLOG`, `GRKERNSEC_SIGNAL` však ano.

Grsecurity je schopno povýšit GNU Linux na úroveň C2 v klasifikaci TCSEC. Je možno pomýšlet také na klasifikaci B1?

Třída B1 přepokládá

splnění požadavků třídy C2 ,
GNU Linux bez dodatečných úprav nesplňuje. S patchem Grsec s příslušnými volbami však ano – viz výše.
klasifikaci subjektů a objektů podle stupně jejich ochrany,
GNU Linux splňuje – viz kapitola 5.3.2.
povinné řízení přístupu (MAC) a
GNU Linux bez dodatečných úprav nesplňuje. S patchem Grsec s příslušnými volbami však ano – viz kapitola 9.18.
existenci neformální definice bezpečnostní politiky.
GNU Linux bez dodatečných úprav splňuje, neboť poskytuje definici formální

Grsecurity je schopno povýšit GNU Linux na úroveň B1 v klasifikaci TCSEC. Je možno pomýšlet také na klasifikaci B2?

Třída B2 předpokládá

splnění požadavků třídy B1 ,
GNU Linux bez dodatečných úprav nesplňuje. S patchem Grsec s příslušnými volbami však ano – viz výše.
formální definici modelu bezpečnostní politiky IS,
Formální definice modelu bezpečnostní politiky Grsec existuje. Je jí dokument Grsecurity ACL documentation [Spe03]. Formální definice modelu bezpečnostní politiky linuxu je umístěna v archivu, v němž je distribuováno jádro samotné v adresáři `Documentation`.
MAC nad všemi subjekty a objekty v systému a
GNU Linux bez dodatečných úprav nesplňuje. S patchem Grsec s příslušnými volbami však ano – viz kapitola 9.18.
dekompozici IS do modulů.
GNU Linux splňuje – viz např. kapitoly 4.3 či 5.1.1.

Grsecurity je schopno povýšit GNU Linux na úroveň B2 v klasifikaci TCSEC. Je možno pomýšlet také na klasifikaci B3 ?

Třída B3 předpokládá

splnění požadavků třídy B2 ,
GNU Linux bez dodatečných úprav nesplňuje. S patchem Grsec s příslušnými volbami však ano – viz výše.
přenechává jakékoliv autorizace referečním monitorům,
GNU Linux bez dodatečných úprav nesplňuje. S patchem Grsec také ne. Referenční monitor zprostředkovávající autorizaci všech přístupů subjektů k objektům ani aplikace patche Grsec nezavádí.
definuje roli bezpečnostního správce.
GNU Linux bez dodatečných úprav nesplňuje. S patchem Grsec také ne. Role administrátora zabezpečení GNU Linux, byť obohacený patchem Grsec, nezná.

Aplikací patche Grsecurity lze povýšit GNU Linux z úrovně C1 [5] na úroveň B2 dle klasifikace TCSEC. **Na vyšší ohodnocení patch Grsec nestačí** z důvodu absence institutu referenčních monitorů a role bezpečnostního správce v systému.

[5] Úroveň C1 dle TCSEC GNU Linux dosahuje bez dalších úprav [Pat].

15. Závěr

Aby bylo možno na závěr určit míru dosaženého Zvýšení bezpečnosti Linuxu, bylo zapotřebí nejprve čtenáře seznámit s obecně uznávanou klasifikaci bezpečnosti informačních systémů TCSEC (viz kapitola 2). GNU Linux bez dodatečných úprav tato klasifikace bezpečnosti zařazuje do třídy C1[1]. Dále byly rozebrány mechanismy správy paměti. Příčinou toho bylo očekávání, že stěžejní bezpečnostní vylepšení se odehrají právě na tomto poli. Toto očekávání se beze zbytku naplnilo. Machinace s pamětí, jsou podstatou existence většiny hrozeb[2]. Proto se vylepšení zabezpečení práce s pamětí týká i většina bezpečnostních technik v tomto dokumentu uvedených. Dále byla zmíněna existence v GNU Linux dnes již standardně přítomných bezpečnostních mechanismů (viz kapitola 5).

K docílení pozorovatelného Zvýšení bezpečnosti Linuxu bylo zapotřebí vybrat řešení, na kterém bude toto zvýšení postaveno. Do užšího výběru se dostala trojice SELinux, AppArmor a Grsecurity (viz kapitola 7). Byly analyzovány možnosti, které tato řešení nabízí. Z jejich následného srovnání bylo vybráno řešení pro Zvýšení bezpečnosti Linuxu nejvhodnější – Grsecurity. To své konkurenty překonalo především rozsahem své působnosti. Neomezuje se totiž zdaleka jen na řízení přístupu k objektům ležícím na FS, nýbrž vnáší do Linuxu celou řadu úprav týkajících se všemožných oblastí, ve kterých se jádro angažuje počínaje modifikací implementace síťového zásobníku (viz kapitola 9.19) přes pečlivější izolaci chroot prostředí (viz kapitola 9.17) až po rozsáhlé zásahy bezpečnostních technik do mechanismů správy paměti.

Aby nebylo Zvýšení bezpečnosti Linuxu čtenáři vnucováno jen tak, bez příčiny, věnoval se autor i hrozbám, které jsou důsledkem zranitelností, které mohou být nejen v jádře, ale také, a to zejména, v uživatelských programech. Vzhledem k tomu, že jádro je jediným procesem běžícím v systému s úrovní oprávnění CPL vyšším nežli 3, je jádro jediným procesem, který má oprávnění vykonávat dohled nad ostatními procesy a proto za jejich nekalé chování svým způsobem nese zodpovědnost. Jádro představuje arbitra, který v systému ručí za dodržování pravidel hry. Od jádra se očekává, že bude za všech okolností vymáhat dodržování správcem systému definovaných přístupových oprávnění. Bezpečnostní techniky aplikované na jádře se proto často netýkají odstranění zranitelnosti v jádře, nýbrž prevence zneužití zranitelností v uživatelských programech. Cílem aplikace těchto bezpečnostních technik je dosažení stavu, kdy provoz zranitelnostmi naditých programů nebude pro systém jako takový znamenat hrozbu. Jádro ve své výchozí podobě je však v boji proti hrozbám všeho druhu buďto zcela vůbec či spíše jen nedostatečně vybaveno.

Toto se snaží napravit záplata Grsec obohacením jádra o různé bezpečnostní mechanismy. Jejich popisu byla věnována samostatná kapitola 9. Dopad na úroveň zabezpečení Linuxu byl tímto čtenáři odhalen. Nepřekvapivě se Grsec ukázal býti přínosem ve všech disciplínách, kterým se věnuje. V mnohých z nich však vanilla Linux není za GrsecGrsecem tak pozadu, jak by se mohlo zdát (viz kapitola 10). Grsecurity vynucuje povinnou kontrolu NX bitu při každém instrukčním přístupu do jakéhokoliv mapování ve VAS procesu. Na architektuře x86-32, která NX bit nezná, jej Grsec emuluje, aby ochranu před útoky typu „shellcode injection“ dostala i na ni. Neupravený LinuxLinux použití NX bitu vynucuje jen pro vybraná mapování (více viz 10), což zabrání některým útokům, nicméně rozhodně ne všem. Grsec přináší kompletní ochranu před útoky ret2usr pro všechny verze Linuxu v podobě bezpečnostních technik KERNEXEC a UDEREF. Systém provozující vanilla Linux je před tímto útokem chráněn také, avšak ne vždy[3]. Obecně lze říci, že Grsec přináší zpravidla maximální v dané době dostupné zabezpečení za všech okolností. Linux nějakou formou tohoto zabezpečení disponuje také. Nebývá však tak propracovaná, přichází později a zpravidla jen do nejnovější verze jádra. Vanilla Linux proto nelze nazvat obecně nezabezpečeným, snad jen „vůči Grsec opožděným“.

Pro vyslovení závěrečného doporučení bylo zapotřebí kromě dopadu na úroveň zabezpečení zjistit též dopad Grsecu na výkonnost jím zabezpečovaného systému. Byla proto vytvořena množina funkcí tvořících společně prostředí pro automatizované získávání hodnot, jejich následné postupné zpracování a export do formací nakonec importovaných do tohoto dokumentu. Těmto funkcím se detailněji věnuje kapitola 12. Tyto funkce byly následně provozovány v měřícím/měřeném prostředí, jehož příprava spočívala především v kompilaci různých variant téže verze[4] jádra. Každé jádro mělo aktivovánu jednu vybranou volbu, aby mohl být změřen výhradně její dopad na výkonnost systému.

Nesnáze vyvstalé během tvorby dokumentu v podobě nekompatibility většiny z již zkompilovaných jader s virtualizačním prostředím Xen[5] či v podobě nutnosti rekompilace a přeměření[6] některých z jader až ve fázi psaní hodnocení

[1] Třída C1 představuje „druhé nejhorší ohodnocení“ v rámci klasifikace TCSEC. Systémy, kterým je přiděleno, povinně používají volitelnou ochranu – DAC – i autentizaci všech subjektů. Více viz kapitola 2.2.1.2.

[2] Jsou jimi např. shellcode injection, return-to-lib, Return to userspace či RAP.

[3] Ochrana před útokem ret2usr je podmíněna použitím Linuxu alespoň ve verzi 3.7 a použitím procesoru Intel Haswell či novějším.

[4] Pro měření bylo zvoleno jádro verze 4.8.16. Původním záměrem bylo vzít distribuční jádro Debianu 9 – Linux 3.16, avšak to kladlo aplikaci patche Grsec zvýšený odpor.

měření byly zdárně překonány.

Měření probíhala automatizovaně do té míry, že pro každou jednu konfiguraci měřícího/měřeného prostředí – jádro s aktivovanou volbou – byla spuštěna testovací funkce, která pak běžela autonomně do té doby, dokud nedoběhl poslední test, který měla vykonat. Měřeními byl prokázán negativní vliv Grsec na výkonnost systému, avšak zdaleka ne takový, jaký autor očekával. Na počátku studia patche Grsec/PaX autor nabyl dojmu[7], že stěžejní úlohou Grsec je úplné zprovoznění NX bitu na architektuře x86-64 a jeho emulace na architektuře x86-32. Očekával, že výkonnostní propad se dostaví zejména v případě bezpečnostních technik emulujících NX bit. Testy však ukázaly, že jeho předpoklad byl chybný a aktuálně výkonností hýbou jiné bezpečnostní techniky. Měřením bylo zjištěno (viz kapitola 14.1), že největší redukci výkonnosti v rámci Grsec způsobují volby `PAX_MEMORY_UDEREF`, `PAX_MEMORY_STACKLEAK` a `PAX_MEMORY_SANITIZE`. Každá z nich poskytuje pokročilou techniku ochrany před zneužitím určité zranitelnosti. Dvě z těchto se věnují zabezpečení jádra a třetí likvidaci obsahu paměti po jejím použití. V kapitole 8.9 zmíněný trend stoupajícího podílu útoků směřujících na jádro na úkor uživatelských procesů koresponduje s výpočetní náročností ochranných mechanismů proti těmto útokům.

Dopad zvoleného řešení – Grsec – na úroveň zabezpečení Linuxu (a potažmo tak i GNU/Linuxu) vně i v prostředí CIV ZČU je pozitivní. Aplikací Grsecurity lze eliminovat drtivou většinu hrozeb v tomto dokumentu zmiňovaných. GNU Linux může být povýšen Grsecurity z úrovně C1 až na úroveň B2 dle klasifikace TCSEC (viz předchozí kapitola). Cenou za aplikaci bezpečnostních technik z Grsecurity je, s výjimkou několika málo z nich, jen nepatrný pokles výkonnosti systému. Doporučení autora uvedená v kapitole 14.2 jsou proto celkem jednoznačná. Autor doporučuje aktivaci většiny nadstandardních bezpečnostních technik ve většině nasazeních. Vzhledem k obecné rozšířenosti technik útoků, jakými jsou např. shellcode injection či return-to-lib, je namístě považovat důsledné vynucení nespustitelnosti mapování přítomných v userspace a znahodňování počátků mapování již za standardní bezpečnostní techniky. Provozovatel systému, který dbá na jeho zabezpečení, by se proto měl porozhlédnout minimálně po bezpečnostních technikách `PAX_NOEXEC` a `PAX_ASLR` či jejich alternativách z řad konkurence. Jejich dopad na výkon je dnes již zanedbatelný a přírůstek zabezpečení značný, soudí autor. Existuje další nespočet voleb v rámci patche Grsec/PaX, jejichž aktivace znamená pro systém mnohem větší přínos[8] nežli jakou přináší systému ztrátu[9] – z testovaných lze jmenovat například volby `NO_SIMULT_CONNECT` či `BLACKHOLE`. Za zmínku stojí, že tyto nepůsobí preventivně jako většina ostatních zde zmiňovaných, nýbrž v zájmu bezpečnosti naprosto cíleně pokřivují síťový zásobník předpisům navzdory.

Významný výkonností dopad byl zjištěn v podstatě jen u voleb `PAX_MEMORY_SANITIZE`, `PAX_MEMORY_STACKLEAK` a zejména pak u volby `PAX_MEMORY_UDEREF`. Tyto volby díky své vyšší náročnosti prozatím zůstávají v pásmu nadstandardních bezpečnostních technik. Vyšší náročnost je dána především jejich charakterem – působí totiž čistě preventivně. Jejich zapnutí ponechává autor na zvážení provozovatele systému. Obecně však lze jejich zapnutí doporučit na počítačích provozujících terminálové, případně pak i webové servery, neb právě takové počítače jsou nejvíce ohroženy. Jsou na nich totiž provozovány často neověřené, rychle psané programy, které, jak již bylo řečeno, obsahují zranitelnosti.

V době zahájení prací na tomto dokumentu bylo Grsec dostupné zdarma ve stabilní verzi. Tento stav se, bohužel, změnil (viz kapitola 6.6.1). Pro CIV ZČU byla ke dni 17. 5. 2017 poskytnuta nabídka – licence Grsec pro 100 počítačů[10] v přepočtu za 119040 Kč/rok. Zpoplatnění Grsecu není zrovna radostnou zprávou, avšak nikoliv také katastrofou. Dobrovolné vyklizení pozic ze strany Grsecurity totiž pravděpodobně povede k zvýšené poptávce po integraci nových bezpečnostních technik směrované přímo k vývojářům jádra. Pro úroveň zabezpečení jádra to nakonec může vyznít i pozitivně v tom smyslu, že bezpečnostní techniky se tak do něj nakonec dostanou rychleji a ve větším množství.

Doporučení podaná čtenáři ohledně jednotlivých voleb v kapitole 14.2 pochopitelně vedou ke zvýšení zabezpečení systému. Avšak tak, jako i bezpečnostní řešení, tak i schopnosti osob bez pověření – útočníků – postupují neustále kupředu a nelze proto s jistotou tvrdit, že systém se po nasazení něčeho stane bezpečným. Nestane. Pouze se stane relativně bezpečnějším vzhledem k systémům běžně provozovaným v kontextu jeho doby.

[5] Testovací prostředí bylo původně částečně provozováno virtualizovaně. Konkrétně počítač xenmv byl virtualizován. Bylo tomu tak z důvodu snadnější nápravy případného selhání při výměně jader, která během měření probíhala takřka neustále. Avšak autorovi se nepodařilo Grsecurity přesvědčit k fungování v takovém prostředí a jádra jím obohacená odmítala bootovat. Finální podoba testovacího prostředí, kdy je xenmv fyzickým strojem je tak pouze náhradním, avšak funkčním, řešením.

[6] K rekompilaci a přeměření vedly smysl nedávající výsledky některých měření. Více viz kapitola 11.2.

[7] Značný podíl zdrojů, týkajících se Grsec/PaX, které autor nalezl, se věnoval právě NX bitu a jeho emulaci. Zřejmě kvůli všeobecné známosti této problematiky.

[8] Přínos volby bývá zpravidla jen v oblasti zabezpečení, nicméně najdou se i takové volby, které mají pozitivní dopad také na výkonnost systému – kupříkladu volba `BLACKHOLE`, což dokazují testy 13.6.1 či 13.7.1.

[9] Ztráta volbou zapříčiněná je cenou za její přínos a týká se zpravidla výkonnosti systému. Výjimkou mohou být ztráty ve funkčnosti systému. Například volba NOEXEC znemožňuje provoz programu Matlab, neboť ten pro svůj chod vyžaduje spouštění jím generovaného strojového kódu, který si ukládá na zásobník, jehož obsah je však při aktivaci volby NOEXEC striktně nespustitelný.

[10] Tato nabídka je podmíněna provozem výhradně nekomerčního/vzdělávacího software na zabezpečovaných počítačích.

A. Seznamy

A.1 Obrázky

A.2 Tabulky

A.3 Příkaz/y (interakce s shellem)

A.4 Skripty (posloupnosti příkazů)

A.5 Konfigurace GrSec

A.6 Strojové kódy x86

A.7 C kódy

A.8 Konfigurace

A.9 Výrazy

A.10 Rovnosti

A.11 Zkratky

AC Access Control 38, 40, 44, 48, 49, 51–53, 55, 57, 67, 86, 91, 205, 206, 213, 215, 218
ACL Access Control List 16, 38–40, 44, 47, 51, 85, 94, 212–214, Slovník: Access Control List
ANSI C American National Standards Institute C 134
API Application Programming Interface Slovník: Application Programming Interface
ARMv6 ARMv6 75
AS Address Space 9, 23, 25, 26, 28–30, 37, 57–60, 67, 68, 70, 73–75, 77, 94, 95, 101, 213–219, Slovník: Address Space
ASLR Address Space Layout Randomization 9, 55, 61, 63, 67–74, 84, 89, 95, 206
AZ aktivační záznam 29–34, 58–60, 62, 78, 216, 217, Slovník: aktivační záznam

BSS Better Save Space Slovník: BSS

Chroot Change Root 55
CIV Centrum informatizace a výpočetní techniky 5, 13, 54, 55, 67, 94, 204, 206, Slovník: CIV
CPL Current Privilege Level 96, 203, 213, Slovník: Current Privilege Level
CPU Central Processing Unit 74, 76
CVSS Common Vulnerability Scoring System Slovník: Common Vulnerability Scoring System

DAC Discretionary Access Control 9, 13, 16, 38–41, 47–49, 51, 95, 201, 203, 213, Slovník: Discretionary Access Control
DBMS Database Management System 197, Slovník: Database Management System
DoS Denial of service 65, 92
DTLB Datová TLB 76

EFA Extended File Attributes 52, 53, Slovník: Extended file attributes
EGID Effective GID 20, 21, 39–41
EI Effective Identity 20, 21, 38–41, 44, 47, 218, Slovník: Effective Identity
ELF Executable and Linking Format 60, 63, 68, 73, 78, 79, 81, 82, 214, Slovník: Executable and Linking Format
EUID Effective UID 20, 21, 39, 40

FS filesystem 28, 29, 48, 49, 52, 57, 60, 79, 84, 85, 88, 89, 103, 107, 109, 110, 112, 114, 161, 203, 205, 215

GDT Global Descriptor Table 80, 215, Slovník: Global Descriptor Table
GID Group Identifier 19–21, 39–41, 92–94, 210–213, 217
GNU GPL GNU General Public License 215, Slovník: GNU General Public License
GRP-O GrSec politika vztažená k objektu 87, 88, 90, 91, 206
GRP-S GrSec politika vztažená k subjektu 87–91, 206, 209
GRP Grsec politika 87–91, 209, Slovník: Grsec politika
GrSec GrSecurity 10, 84, 86, 88–94, 97, 98, 100, 206, 209
Grsec Grsecurity 10, 23, 42, 43, 54, 55, 67, 79, 81, 84–89, 91–96, 100, 101, 117, 135, 139, 148, 156, 164, 166, 169, 171, 174, 179, 182, 185, 198, 201–204, 206, 207, 209, 211, 217
Grsecurity ACL Grsecurity Access Control Lists 85, 199

HM High Memory 24
HTT Hyper-Threading Technology Slovník: HTT

ICMP Internet Control Message Protocol 92, 115, 117
IDS Intrusion Detection System 44, Slovník: Intrusion Detection System
IDT Interrupt Descriptor Table 80, Slovník: Interrupt Descriptor Table
Intel MPX 13, Slovník: Intel Memory Protection Extensions
IP Internet Protocol 43, 86, 89, 117
IPC Inter-Process Communication 84, Slovník: Inter-Process Communication
IS informační systém 15–17, 44, 201, 202, 213, 215
ITLB Instrukční TLB 27, 75, 76

Jail Jail Chroot Project 43

KM-NPC kernel mode - Not Process Context 21
KM-PC kernel mode - Process Context 21, 24
KM Kernel Mode 21, 24, 215, Slovník: kernel mode
KS Kernel Space 24, Slovník: Kernel Space

LA Logical Address 23, 24, 26, 76, 77
LAS Logical Address Space 23–26, 76, 215, 218, Slovník: Logical Address Space
LDT Local Descriptor Table 215, Slovník: Global Descriptor Table
LIDS Linux Intrusion Detection System 44, Slovník: Linux Intrusion Detection System
LM Low Memory 21, 24
LSM Linux Security Modules 48, 49, 51, 54, 55, 205, 218

MAC Mandatory Access Control 9, 10, 16, 17, 38, 47–49, 51, 52, 54, 55, 67, 85–91, 199, 202, 206, 210, 213, 215, Slovník: Mandatory Access Control
metalist metalist.civ.zcu.cz 10, 97, 114, 120, 129, 131, 132, 216
MMS Memory Mapping Segment 28, 29, Slovník: MMS
MMS Memory Mapping Segment Slovník: MMS
MMU Memory Management Unit 25–27, 37, 75, 76, 95, Slovník: Memory Management Unit

NA Návratová adresa 31, 35, 60, 61, 83, 217
NMMP Nestálost měřícího/měřeného prostředí 117, 123, 125, 126, 139, 158, 216, Slovník: Nestálost měřícího/měřeného prostředí
NX No eXecute bit 55, 67, 74, 75, 82, 96, 97, 203, 204, Slovník: No eXecute bit
NZPV nezávisle proměnná veličina 115, 118, 120, 134, 137, 140, 142, 145, 154, 156, 159, 161, 162, 164, 167, 169, 172, 174, 177, 180, 182, Slovník: nezávisle proměnná veličina

OFP Old Frame Pointer 30, 33, Slovník: Old Frame Pointer
OS operační systém 15, 38, 215, 218

PA physical address 23–27, 76, 215, 216, 218
PAS Physical Address Space 23–28, 216–218, Slovník: Physical Address Space
PaX Page eXec project 9, 10, 23, 37, 55, 67–95, 100–102, 198, 201, 204, 206, 207, 209, 210, 216, 218, Slovník: Page eXec project
PC Program Counter Slovník: čítač instrukcí
PC Program Counter Slovník: čítač instrukcí
PGID Primary GID 19, 21
PIC Position Independent Code 68, 81, 216, Slovník: Position Independent Code

PID Process Identifier 20, 61, 89, 91
PIE Position Independent Executable 68, 69, 73, 209, 214
POSIX ACL POSIX 13
POSIX Portable Operating System Interface 212, Slovník: POSIX
PT Page Table 27, Slovník: tabulka stránek
PTBR Page-table base register 27, Slovník: Page-table base register
PTLR Page-table length register Slovník: Page-table length register

RAP Return Address Protection 10, 13, 83, 84, 96, 201, 203, 209, 217, Slovník: Return Address Protection
RBAC Role Based Access Control 9, 10, 38, 47, 51, 55, 67, 85–91, 95, 206
ret2dir Return to direct mapped memory 64, 67, 79, 80, 214, Slovník: Return to direct mapped memory
ret2usr Return to userspace 10, 64, 79, 80, 96, 201, 203, Slovník: Return to userspace
RFC Request for Comments 96, Slovník: Request for Comments
RGID Real GID 20
RI Real Identity 20, 21, 41, Slovník: Real Identity
RIP Return Instruction Pointer 29–31, 33, 34, 59, Slovník: Return Instruction Pointer
RJ rozsahová jednotka 29, 32–34, 59, 60, 78, Slovník: rozsahová jednotka
RM Reference Monitor 17, Slovník: Reference Monitor
ROP Return-Oriented Programming 9, 62, 64, 83, 84, 96, 201, 217, Slovník: Return-Oriented Programming
RP RPLONG see also Mandatory Access Control
RUID Real UID 20, 21

SB supervisor bit 76, Slovník: supervisor bit
SE-DS datový segment SEGMEXEC 77, 78
SE-KS kódový segment SEGMEXEC 77, 78
SELinux Security-Enhanced Linux 48, 98
SFP Saved Frame Pointer 29–31, 33, 34, 59, Slovník: Saved Frame Pointer
SGID Set GID 21, 85
SMAP Supervisor Mode Access Protection 13, 79, 80, 96
SMEP Supervisor Mode Execution Protection 13, 79, 80, 96
SMT Simultaneous Multithreading 97, 214, Slovník: SMT
SSUID Saved SUID 20, 21
SUID Set UID 21, 44, 85, 94, 212
SWAP odkládací prostor ve vnější paměti 26, Slovník: odkládací prostor

TCB Trusted Computing Base Slovník: Trusted Computing Base
TCP Transmission Control Protocol 64, 65, 92, 96, 117, 118, 120, 122, 123, 125, 126, 128, 129, 131, 132, 134, 135, 199, 205
TCSEC Trusted Computer System Evaluation Criteria 7, 9, 15–18, 201–204, Slovník: TCSEC
TDI Trusted Database Interpretation 15, Slovník: TDI
TE Type Enforcement Slovník: Type Enforcement
TLB Translation Lookaside Buffer 25, 27, 75, 76, 210, 211, 214, Slovník: Translation Lookaside Buffer
TNI Trusted Network Interpretation 15, Slovník: TNI
TPE Trusted Path Execution 87, Slovník: Trusted Path Execution
TS Trusted System Slovník: Trusted System
TSS Task State Segment 218, Slovník: Task State Segment

UDP User Datagram Protocol 92, 117, 123, 125, 126, 128, 129, 131, 133, 134
UID User Identifier 19–21, 39, 40, 210, 212, 213, 217
UM User mode 21, 24, 215, 219, Slovník: user mode
US User Space 24, 77, 80, 84, 85, 87, 94, Slovník: User Space
USA United States of America 15

VA virtual address 26, 27, 76, 77, 215, 218
VAS Virtual Address Space 20, 21, 23, 24, 26–29, 60–62, 67, 73, 76, 78, 81, 89, 203, 213, 215, 217–219, Slovník: Virtual Address Space

x86-32 x86 (IBM PC) s šířkou adresové sběrnice procesoru 32 b 24, 26, 37, 67, 77, 80, 82–84, 95–98, 203, 204, 209, 216
x86-64 x86 (IBM PC) s šířkou adresové sběrnice procesoru 64 b 32, 64, 73–75, 80, 82, 95, 96, 152, 204, 209, 216, 217
x86 x86 10, 23, 26–28, 30–32, 34, 37, 62, 63, 75, 77, 80, 83, 84, 95, 125, 209, 212, 213, 217–219, Slovník: x86
xenmv xenmv.civ.zcu.cz 10, 97, 114, 118, 120, 125, 129, 131, 132, 142, 204, 209, 216
XPFO eXclusive Page Frame Ownership 64, 80, 96, Slovník: eXclusive Page Frame Ownership

ZPV závisle proměnná veličina 115, 118, 120, 137, 140, 142, 145, 154, 156, 159, 162, 164, 167, 169, 172, 174, 177, 180, 182, Slovník: závisle proměnná veličina

ZČU Západočeská univerzita v Plzni 13, 54, 94, 204, 206, 213

A.12 Slovník

Access Control List Seznam oprávnění připojený k právě jednomu objektu. Obsahuje výčtem definovaná oprávnění uživatelů či uživatelských skupin nakládat s daným objektem 16, 38–40, 44, 47, 51, 85, 94, 210, 213, 214

Address Space Adresní prostor vyhrazený pro jeden proces. Obecně je jím myšlen zpravidla VAS. 9, 23, 25, 26, 28–30, 37, 57–60, 67, 68, 70, 73–75, 77, 94, 95, 101, 210, 213–219

Address Space Layout Randomization Technika zajišťující dynamické mapování jednotlivých modulů procesu po VAS. 58, 61, 67, 73, 217

aktivační záznam Datová struktura, která představuje lokální prostředí výpočtu – tzv. kontext – v němž se nachází vlákno při vstupu do rozsahové jednotky. 29–34, 58–60, 62, 78, 206, 210, 216, 217

aktivační záznam Datová struktura, která představuje lokální prostředí výpočtu – tzv. kontext – v němž se nachází vlákno při vstupu do rozsahové jednotky. 29, 30

anonymní mapování Mapování, jehož předobrazem není soubor, nýbrž vzniklo až na základě běhu programu. Typicky se jedná o zásobník a haldu. 29, 37, 73–75, 156

AppArmor Modul Linuxu poskytující vlastní implementaci MAC. 51, 55, 95, 203, 210

AppArmor profil Textový soubor obsahující nastavení přístupových oprávnění pro konkrétní množinu aplikací. 51

Application Programming Interface Rozhraní pro programování aplikací. Sbírka procedur, funkcí, tříd či protokolů nějaké knihovny (ale třeba i jiného programu nebo jádra operačního systému), které může programátor využívat. 210

BSS (mapování) Mapování v rámci AS procesu, do něhož se umisťují neinicializované statické proměnné. Více viz sekce 4.6.3. 29, 58, 210, 216

buffer Paměťový prostor vyhrazený pro zápis proměnné 213

buffer overflow Zápis dat do bufferu nedostatečné velikosti. 57–59, 214

capabilities Mechanismus řízení přístupu v rámci GNU Linux, které umožňuje delegovat přístupová oprávnění dříve exkluzivní pro superuživatele jiným uživatelům. 13, 20, 43, 44, 88, 93, 200

chroot Systémové volání, které změní kořenový adresář procesu a všech jeho případných potomků vzniklých voláním `fork()`. 42, 63

chroot prostředí Podstrom souborového systému vzniklý voláním `chroot()` za účelem běhu vybraných procesů v izolovaném prostředí. 9, 10, 42, 43, 55, 62, 63, 67, 84, 85, 203, 206, 215

CIV Centrum informatizace a výpočetní techniky je pracovištěm v rámci ZČU poskytující jí služby týkající se výpočetní techniky. 5, 13, 54, 55, 67, 94, 204, 206, 210

Common Vulnerability Scoring System Průmyslový standard pro posuzování závažnosti bezpečnostních zranitelností IS. Jednotlivým zranitelnostem přiřazuje skóre v intervalu $\langle 0, 10 \rangle$ dle stupně jejich závažnosti. 10 značí nejvíce závažnou zranitelnost. Vstupními parametry pro výpočet skóre je několik metrik klasifikujících snadnost zneužití zranitelnosti a potencionální dopad zneužití zranitelnosti. 210

Current Privilege Level Architektura x86 disponuje čtyřmi úrovněmi oprávnění CPL. Jedná se o hodnotu představující aktuální úroveň oprávnění přidělenou právě prováděnému procesu. Ukládá se do selektoru kódového segmentu. 96, 203, 210, 213

data (mapování) Mapování v rámci AS procesu, do něhož se umisťují statické proměnné inicializované ve zdrojovém kódu programu – globální proměnné a lokální „static“ proměnné. Více viz kapitola 4.6.2. 28, 29, 58, 216

Database Management System Softwarové vybavení, které zajišťuje práci s databází, tzn. tvoří rozhraní mezi aplikačními programy a uloženými daty. 197, 210

dentry Objekt představující a odkazující se na jednu položku adresáře, respektive na její inode. Je ukládán do datových bloků souboru představujícího adresář. 94

Discretionary Access Control Nepovinné (⊠olitelné) řízení přístupu. Řízení přístupu založené na existenci ACL pro každý objektu v systému. Definici ACL má v moci vždy vlastník příslušného objektu. Detailněji se o DAC zmiňuje sekce 9, 13, 16, 38–41, 47–49, 51, 95, 201, 203, 210, 213

do_page_fault() Funkce obsluhující výjimku Page fault. Modifikací této funkce lze implementovat vlastní AC do paměti. 73

doména Oblast/obor působnosti. 16, 38

dopad Škoda vzniklá v důsledku působení hrozby. 57

dynamický linker Část operačního systému, která načítá z externí paměti do hlavní paměti a propojuje sdílené knihovny se spustitelným programem za běhu programu. 70

Effective Identity Uspořádaná dvojice (UID,GID) identifikující uživatele, pod jehož oprávněními proces běží. Tato identita je používána k autorizaci procesu uvnitř systému. 39

EMULTRAP Technika, která použití trampolín umožní i v případě aktivního PAGEEXEC/SEGMEXEC [GrSecDoc]. 78, 82

epilog Posloupnost instrukcí vykonávaná při návratu z těla funkce. 62, 63, 83, 84, 209, 217

etdyn Jiné pojmenování pro binární soubor ve formátu PIE. 73, 101

etexec Jiné pojmenování pro binární soubor ve formátu ELF. 73, 101

eXclusive Page Frame Ownership Technika ochrany proti útokům typu ret2dir. Více viz kapitola 9.5.4. 64, 80, 96, 212

Executable and Linking Format ELF je standardizovaný typ spustitelného souboru vyskytující se v prostředí GNU Linux. 60, 63, 68, 73, 78, 79, 81, 82, 210, 214

exploit Sekvence příkazů, které využívají zranitelnost k provedení původně nezamýšlené činnosti typicky za účelem získání prospěchu útočníka. 57–59, 62, 73, 84, 214, 217

exploitování Aplikace exploitu za účelem zneužití zranitelnosti. 13, 59, 63, 94

Extended file attributes Rozšiřující vlastnost systému souborů, která umožňuje k souborům přidat specifická metadata, která neslouží samotnému systému souborů, nýbrž nesou doplňující informace pro jádro systému, procesy a aplikace. 52, 53, 210

externí paměť Paměť sloužící k dlouhodobějšímu uchování dat. Typicky se jedná o paměť non-volatilní, čili nezávislou na nepřetržitém napájení. 24, 139, 142

Global Descriptor Table Úložiště pro adresy segmentů s globální platností. K nim přistupují všechny procesy čili využívá je celý systém. 80, 211, 215

GNU General Public License Licence pro svobodný software původně napsaná Richardem Stallmanem pro projekt GNU. Vyžaduje, aby od díla odvozená díla byla dostupná pod toutéž licencí 211, 215

GNU Linux Operační systém využívající jádro Linux [Sta] 7, 10, 13, 16, 23, 37, 39, 77, 95, 96, 201–204, 207, 213–215, 217

GPL Licence pro svobodný software vyžadující, aby byla odvozená díla dostupná pod toutéž licencí. V rámci této filosofie je řečeno, že poskytuje uživatelům počítačového programu práva svobodného softwaru a používá copyleft k zajištění, aby byly tyto svobody ochráněny, i když je dílo změněno nebo k něčemu přidáno. 54

Grsec politika Záznam v GrSecurity ACL čili definice přístupových oprávnění **právě jednoho** subjektu v systému. 87–91, 209, 211

Grsecurity Soubor modifikací jádra posilujících bezpečnost distribuovaný formou patche. 9, 23, 54, 55, 67–94, 120, 123, 150, 152, 158, 202–204, 206

GrSecurity ACL Procesně-orientovaná ACL, která pracují s objekty, subjekty a rolemi. 81, 86, 87, 91, 214

halda (mapování) Mapování v rámci AS procesu, do něhož se umisťují proměnné, kterým nemůže být přidělen prostor již při překladu. Více viz sekce 4.6.4. 28, 29, 58, 61, 74, 95, 217

Hlavní paměť Jedna z vnitřních pamětí počítače. Paměť sloužící pro uchování momentálně zpracovávaných dat a programů. Typicky se jedná o paměť volatilní, čili závislou na nepřetržitém napájení. 23, 24, 80, 121, 123, 125, 142, 144

hrozba Potenciální příčina nechtěného incidentu, která může vyústit v poškození systému. 7, 13, 42, 57, 58, 84, 203, 213, 217, 219

HTT Hyper-Threading Technology je obchodní název pro SMT, kterou zavedla u svých procesorů společnost Intel. 211

i586 Rodina x86-kompatibilních procesorů Intel vydaných v roce 1993 pod názvem Pentium. Jednalo se o první x86 procesory, které disponovaly dvěma TLB. 75

IA-32 IA-32 – Intel Architecture, 32bit – je v informatice označení architektury 32bitových CISC procesorů, které se používají zejména v počítačích PC/AT kompatibilních. Lze se též setkat s označením „x86-32“. 71, 74, 75, 77, 78, 206, 216, 218

implicitní maska práv Doplněk do neomezených přístupových práv k objektu. Udržuje se pro tři množiny subjektů – vlastníka objektu, skupinového vlastníka objektu a ostatní subjekty. 39, 40

inode Datová struktura uchovávající metadata souboru. Těmito metadaty mohou být čas poslední změny, čas posledního přístupu, přístupová práva a v neposlední řadě též adresy datových bloků nesoucích data. 19, 39, 44, 48, 49, 52, 94, 213, 218

Intel Memory Protection Extensions Rozšíření instrukční sady x86 zavedené v rodině procesorů Intel Skylake [OKB17]. Pro svou funkčnost vyžaduje podporu ze strany operačního systému[1] i kompilátorů. Je-li řeč o kompilátorech, znamená to, že spustitelné soubory musí být přeloženy s podporou této technologie, což dnes běžně nejsou. Při nasazení Intel MPX dochází ke kontrole ukazatelů, jejichž hodnoty by mohly být změněny během běhu programu útokem buffer overflow. 13, 211

Inter-Process Communication Množina technik pro výměnu dat mezi dvěma nebo více procesy nebo thready. 84, 211

[1] Linux techniku *Intel MPX* podporuje od své verze 3.19.

Interrupt Descriptor Table Tabulka popisovačů segmentů obsluhy přerušení. Schraňuje ukazatele na rutiny obsluhující přerušení. 80, 211

Intrusion Detection System Systém detekující neautorizovaný průnik do IS. 44, 211

jail Nástroj pro konstrukci jail prostředí šířený pod licencí GNU General Public License napsaný v jazyce C. 43

jail prostředí Prostředí podobné chroot prostředí s více omezeními. 43, 215

jádro Program, který koordinuje činnost ostatních programů a zprostředkovává jim prostředky počítače. Alternativně: Strojový kód zavedený do paměti po startu počítače běžící v KM mající neomezený přístup ke všem instrukcím procesoru i ke všem zařízením přítomným v počítači. 23, 24, 26, 42, 44, 48, 49, 51, 54, 55, 63, 67, 74, 80, 85, 87, 94, 98, 200, 203, 214, 215, 218

kanárek Součást mechanismu umožňujícího detekci přepsání návratové adresy aktivačního záznamu. 60, 61, 63

Kernel mode Stav procesoru, ve kterém procesor vykonává strojový kód jádra. 21, 24, 28, 37, 63, 64, 67, 74, 76, 80, 154, 211, 215

Kernel Space Adresní rozsah ve VAS nepřístupný strojovému kódu běžícímu v UM – uživatelskému procesu. Podrobněji popsáno v sekci 9, 24, 29, 61, 63, 64, 67, 79, 80, 94, 96, 123, 126, 140, 153, 201, 211, 215, 217

kernelspace-stack Mapování obsažené v AS procesu obsahující dynamickou datovou strukturu nutnou pro fungování rekurzivního volání systémových volání. Více viz kapitola 4.6.6. 9, 28, 29, 61, 62, 67, 74, 83, 96, 126, 149, 158, 199, 200, 216

KERNEXEC KERNEXEC je bezpečnostní technika zavádějící v Kernel Space nespustitelné paměťové stránky. 10, 79, 80, 96, 215

kontext procesu Stav procesu nebo též množina informací nutná k obnovení činnosti procesu poté, co byl přerušen. 24

label based MAC Povinné řízení přístupu založené na značkách, tudíž nezávislé na umístění objektu (souboru) ve FS. Značky jsou uchovávány odděleně od objektů. 48, 55

learning mode Mód, v němž se nástroj jinak vynucující přístupová pravidla tato pravidla učí. Veškeré operace jsou procesu povoleny a zaznamenávány. Neshledá-li na nich při kontrole obsluha nic podezdřelého, potom jsou tyto zaznamenané akce prohlášeny za jediné povolené aktivity daného procesu. 51, 55, 87–89, 91

Linux Označení pro jádro operačního systému GNU Linux [Sta] vyvinuté Linusem Torvaldsem v roce 1991 a do dnešních dob rozvíjené pod licencí GNU GPL 7, 13, 26, 28, 37, 43, 51, 63, 68, 74, 77, 80, 82, 94–97, 202–204, 213–215, 217, 218

Linux Intrusion Detection System Patch Linuxu zvyšující bezpečnost specifický tím, že nastavení jím provedená jsou platná vždy pro všechny subjekty, neboť neuvažuje různé uživatelské identity. Více v kapitole 5.5.3. 44, 211

Linux Security Modules Framework umožňující v Linuxu implementovat různé bezpečnostní politiky kategorie MAC. 48

Local Descriptor Table Obdoba GDT obsahující také adresy segmentů, avšak pouze těch, které jsou specifické pro pro právě běžící proces v systému. Každému procesu v systému náleží jedna LDT. 211, 215

Logical Address Space Adresní prostor tvořený logickými adresami. 23–26, 76, 211, 215, 218

logická adresa Adresa používaná v rámci LAS 76

LSM hook Volání funkce umístěné těla jaderné funkce, ve které má smysl provádět AC. LSM hook nese adresu LSM modulu, pokud jej nasazené AC implementuje. 48–50, 209, 215

LSM modul Funkce implementovaná nasazeným AC, která je v různých situacích volána – vyzývána k zodpovězení otázky, zda povolit přístup. 48, 49, 51, 215, 218

Mandatory Access Control Povinné řízení přístupu. Řízení přístupu založené na existenci centrální definice přístupových oprávnění k objektům, kterou spravuje pro danou činnost pověřený uživatel typicky správce systému. Podrobněji je koncept povinného řízení přístupu popsán v sekci 9, 10, 16, 17, 38, 47–49, 51, 52, 54, 55, 67, 85–91, 199, 202, 206, 210, 211, 213, 215

mapování Označení pro jednu funkčně jednotnou oblast paměti v AS procesu. 28, 29, 57–60, 67, 68, 70, 71, 73–75, 77, 95, 101, 203, 213–219

Memory Management Unit Komponenta procesoru provádějící překlad VA na PA a řídící paměťovou sběrnici procesoru. 25–27, 37, 75, 76, 95, 211

memory manager Komponenta OS zodpovědná za správné fungování hlavní paměti. Více v sekci 4.2.1. 24

Michal Švamberg Vedoucí diplomové práce, respektive konzultant nápomocný při tvorbě dokumentu, jehož je tento text součástí. 5, 54, 55, 97

mimo kontext procesu Mimo kontext procesu se obecně vykonává činnost, která se nevztahuje k žádnému určitému procesu. Takovou činnost vykonává například vlákno plánovače procesů. 24

MMS (mapování) *Memory Mapping Segment* je mapování v rámci AS procesu obsahující dynamicky linkované knihovny. Více viz kapitola 4.6.5. 28, 29, 60, 73, 81, 211, 217

MPROTECT Ochrana paměťových stránek v rámci PaX, která zabrání vytvoření mapování, které by bylo zapisovatelné a spustitelné zároveň. 75, 82, 95, 210

Nestálost měřícího/měřeného prostředí V měřícím/měřeném prostředí, které mimo jiné popisuje kapitola 11, je zahrnut síťový spoj mezi počítači xenmv a metalist. Tento spoj není vyhrazen pro komunikaci výhradně zmíněných dvou počítačů, nýbrž je sdílen vícero počítači, neb je na něm umístěn přepínač připojující xenmv a metalist minimálně k výchozí bráně. Provoz v síti může ovlivnit komunikaci mezi xenmv a metalist. Síťové testy uvedené v tomto dokumentu proto mohou být NMMP postiženy. 117, 123, 125, 126, 139, 158, 211, 216

nezávisle proměnná veličina Proměnná veličina, která se mění nezávisle vzhledem k jiným faktorům. Manipulujeme jí záměrně. [Mec12] 113–115, 118, 120, 134, 137, 140, 142, 145, 154, 156, 159, 161, 162, 164, 167, 169, 172, 174, 177, 180, 182, 211

No eXecute bit Příznak umístěný v každém záznamu tabulky stránek nesoucí informaci o tom, zda-li je obsah umístěný na stránce, k níž se záznam váže, spustitelný. V architektuře x86-32 není přítomen a je nutné jej emulovat softwarově. V x86-64 je již přítomen. 37, 67, 74, 75, 80, 82, 95, 96, 125, 152, 153, 198, 199, 211

NOEXEC Technika v rámci PaX vynucující podporu nespustitelných stránek na jakémkoliv mapování. 58, 63, 74, 75, 78, 95, 201, 204, 217

objekt Zdroj existující v rámci systému. V případě operačního systému je zdroj zpravidla reprezentován souborem mapovaným do souborového systému 15–17, 37–41, 47, 52, 87, 89, 91, 202, 203, 213, 215, 217–219

odkládací prostor Adresní prostor vyhrazený v externí paměti počítače. Jeho úlohou je v procesu stránkování (viz sekce 4.4.1) poskytnout úložiště pro odkládání stránek mimo hlavní paměť čímž je kompenzována omezená velikost hlavní paměti – PAS. 26, 212

offset Relativní adresa čili posun vzhledem k počátku. 26, 27

Old Frame Pointer Předchozí (1 krok starý) ukazatel na začátek nadřazeného AZ. 30, 33, 211

Page eXec project Soubor technik, které aktivně brání možnému zneužití zranitelností v podobě softwarových chyb umožňujících útočníkovi získat neomezený přístup do adresního prostoru napadeného procesu. 9, 10, 23, 26, 37, 55, 67–95, 100–102, 198, 201, 204, 207, 209–211, 216, 218

Page fault Výjimka vzniknuvší v případě, že data, ke kterým je přistupováno, nejsou nahrána v hlavní paměti počítače. Její výskyt aktivuje programovou rutinu zajišťující uvolnění rámce a přenos požadované stránky do uvolněného rámce. 26, 37, 61, 73, 76–78, 80, 206, 213

Page-table base register Pojmenování registru procesoru, ve kterém je uložena PA začátku tabulky stránek v hlavní paměti 27, 212

Page-table length register Pojmenování registru procesoru, ve kterém je uložena délka tabulky stránek. 212

PAGEEXEC Jedna ze dvou technik **emulace ne/spustitelných stránek** pro procesorovou architekturu IA-32, které PaX nabízí. Více viz kapitola 9.2.2. 10, 26, 75, 76, 78, 82, 89, 95, 214

paravirtualizace Technika virtualizace, kdy se prostředí pro virtualizovaný stroj neemuluje úplně, ale část instrukcí se zpracovává přímo prostřednictvím reálného prostředí. To však předpokládá spolupráci virtualizovaného stroje s hostitelem. 21

path based MAC Povinné řízení přístupu založené na cestě k souboru v rámci filesystému. 48, 55

Physical Address Space Adresní prostor, jehož délka odpovídá kapacitě v počítači instalované hlavní paměti [Dud04]. Tvoří jej *fyzické adresy (PA)*. V dnešních systémech je typicky řádově menší nežli logický/virtuální adresní prostor. Adresní prostor, jehož délka odpovídá kapacitě v počítači instalované hlavní paměti [Dud04]. Tvoří jej *fyzické adresy (PA)*. V dnešních systémech je typicky řádově menší nežli logický/virtuální adresní prostor. 23–28, 211, 216–218

Position Independent Code Pozičně nezávislý kód je strojový kód, který je možné vykonat nezávisle na tom, na jaké adrese je v operační paměti umístěn. 68, 81, 211, 216

Position Independent Executable Spustitelný soubor obsahující pozičně nezávislý kód (PIC). 68, 73

POSIX Přenositelné rozhraní pro operační systémy vzniknuvší podle rozhraní systému UNIX. Popisuje základní systémovou funkcionalitu 212

POSIX ACL Rozšíření základních přístupových práv. Umožňují definovat libovolně[2] dlouhé seznamy přístupových oprávnění, přičemž základní přístupová práva zůstávají vždy nezměněna. 39, 41, 48, 95

privilegovaný port Port identifikovaný číslem z intervalu $\langle 1, 1023 \rangle$. 43, 45, 209

program counter Speciální registr v procesoru, jehož účelem je adresovat instrukce strojového kódu v hlavní paměti počítače 20

prolog Posloupnost instrukcí následující po každém volání funkce (po každém vykonání instrukce `call`. V rámci prologu se zakládá aktivační záznam na zásobníku a vyhrazuje prostor pro lokální proměnné. 83, 209, 217

převzetí kontroly Softwarový útok vedený na konkrétní proces běžící v systému za účelem získání jeho identity, respektive z ní plynoucích přístupových oprávnění. 38, 42, 55, 60, 62, 67, 81

RANDEXEC Technika měnící počátek mapování text, data a BSS v rámci AS procesu. 73, 89, 101

RANDKSTACK Technika měnící počátek mapování kernelspace-stack v rámci AS procesu. 74

RANDMMAP Implementace Address Space Layout Randomization, která modifikuje umístění jednotlivých mapování AS procesu. Aplikovatelná na mapování text, halda a MMS. 63, 73, 82, 89, 199

RANDUSTACK Technika měnící počátek mapování userspace-stack v rámci AS procesu. 73, 74

RAP cookie Záloha návratové adresy (NA) v podobě její kopie uložené do registru (na architektuře x86-64 se jedná o registr `r12`). Tato záloha je vytvářena při vykonávání prologu funkce a porovnávána s originálem při vykonávání epilogu funkce. Je stěžejní komponentou bezpečnostní techniky RAP v patchi Grsec. 83, 84

Real Identity Uspořádaná dvojice (UID,GID) identifikující uživatele, který spustil proces. 20, 21, 38–41, 44, 47, 210, 212, 218

Reference Monitor Komponenta systému provádějící řízení přístupu respektive rozhodující o schválení či zamítnutí žádostí subjektů o přístup k objektům [OSSec]. 17, 202, 212

relokace Přemístění vůči původní či předpokládané budoucí pozici. V kontextu tohoto dokumentu je relokací zamýšleno přemístění strojového kódu na na jiné místo v VAS, než pro jaké byl překladačem vytvořen. 25, 81

Request for Comments Uspořádaná posloupnost oficiálních dokumentů, které definují standardy implementace síťových protokolů. Dokument do tohoto souboru vložený je dále neměnitelný/neaktualizovatelný. Změny jsou tak prakticky prováděny výhradně vkládáním nových dokumentů, které zpoza jiného pořadového čísla odkazují na dokument jim předcházející, který prohlašují na neplatný. 96, 212

Return Address Protection Technika obrany proti ROP postavená na obohacení prologů a epilogů všech funkcí o kód ověřující skutečnost, že během vykonávání funkce nebyla na vrchol zásobníku podstrčena jiná návratová adresa. 10, 13, 83, 84, 96, 201, 203, 209, 212, 217

Return Instruction Pointer Adresa paměti, na níž leží instrukce, která se bude vykonávat po opuštění rozsahové jednotky, k níž je aktivační záznam vázán. 29–31, 33, 34, 59, 212

Return to direct mapped memory Vylepšená varianta útoku Return to userspace, která staví na faktu, že ochrany kontrolující přístup z Kernel Space do User Space jsou vázány na tabulku stránek a proto pro přístup do User Space využívá přímé mapování. 64, 67, 79, 80, 96, 201, 212, 214

Return to userspace Útok postavený na principu přepsání funkčního ukazatele v rámci Kernel Space tak, aby směřoval do User Space [KPK14]. Více viz kapitola 8.9.2. 10, 64, 79, 80, 96, 201, 203, 212, 217

Return-Oriented Programming Vylepšení útoku „shellcode injection“, které spočívá v rozdělení shellcode na části. Bylo vynalezeno za účelem obejití ochran zabraňujícím vykonávání obsahu mapování userspace-stack procesu (NOEXEC) [Zah11]. 9, 62, 64, 83, 84, 96, 201, 212, 217

return-to-lib Označení pro kategorii útoků, při nichž se útočník snaží změnit tok instrukcí programu za jiné, v AS programu již existující instrukce. 57, 58, 95, 101, 203, 204

Ring 0 Nejvyšší možná úroveň oprávnění v rámci x86-kompatibilního procesoru zaručující neomezený přístup ke všem zařízením. V této úrovni oprávnění běží jádro operačního systému. 26, 37, 64, 76, 80, 218

Ring 1 Druhá nejvyšší úroveň oprávnění v rámci x86-kompatibilního procesoru. V této úrovni oprávnění běží na systémech s Linuxem buďto hypervizor nebo nic. 37, 76

Ring 2 Druhá nejnižší úroveň oprávnění v rámci x86-kompatibilního procesoru. Tato úroveň oprávnění není při nasazení GNU Linux využívána. 37

Ring 3 Nejnižší úroveň oprávnění v rámci x86-kompatibilního procesoru. V této úrovni běží všechny procesy běžící v systému. Kódu v této úrovni oprávnění není dovoleno přistupovat k paměťovým stránkám jiných AS nežli je jeho vlastní AS čili kód běžící v Ring 3 je vězněm vlastního AS a jakékoliv interakce s zařízeními či AS jiných procesů provádí pomocí jádra. 37, 217

Rings Základní bezpečnostní technika přítomná na x86-kompatibilních procesorech. Jedná se o hierarchii 4 úrovní oprávnění, kde každá úroveň zahrnuje všechna oprávnění vzhledem k ní nižších úrovní. Více viz kapitola 5.1.1. 13

riziko Pravděpodobnost, že dojde k uplatnění hrozby a vznikne škoda. Hrozba je tím pravděpodobnější, čím snadnější je určitou zranitelnost zneužít. 57

role Povolení vykonávat množinu transakcí udělované subjektu správcem systému. 47, 85–87, 91, 209, 214

ROP chain Posloupnost ROP gadgetů tvořících kód exploit. 62

ROP gadget Posloupnost instrukcí zakončená instrukcí `ret` nacházející se ve spustitelné oblasti VAS využívaná technikou ROP k sestavení kódu glsg.exploit. 62, 84, 217

rozsahová jednotka Úsek kódu, kterému přináleží určité lokální prostředí výpočtu (typicky lokální proměnné). Rozsahové jednotce odpovídá záznam na userspace-stack zvaný aktivační záznam. Rozsahovou jednotku tvoří zpravidla blok, procedura či funkce. 29–34, 59, 60, 78, 212, 217

rámec Pojmenování jednotky mapování (bloku dat) mezi VAS a PAS umístěné v rámci PAS 26, 27, 216

Saved Frame Pointer Ukazatel na začátek nadřazeného AZ. 29–31, 33, 34, 59, 212

segment Samostatný adresový prostor existující v rámci VAS. Obsahuje zpravidla jednu oblast z VAS dle členění VAS v kapitole 4.6. 25

segmentace Mechanismus virtualizace paměti. Více v sekci 13, 17, 25, 26, 80

segmentová tabulka Tabulka obsahující informace o segmentech VAS. Více viz sekce 25

SEGMEXEC Jedna ze dvou technik emulace ne/spustitelných stránek pro procesorovou architekturu IA-32, které PaX nabízí. 10, 26, 73, 77, 78, 82, 95, 214

SELinux Patch jádra využívající rozhraní definovaná LSM, respektive jedná se o LSM modul. 51–53, 55, 95, 203, 218

SELinux doména Množina SELinux typů. 52

SELinux identita Obdoba unixové identity (uživatele). Zavádí se pouze pro potřeby AC v rámci SELinuxu. Jedna SELinux identita může být přiřazena více uživatelům. 51, 52, 218

SELinux kontext Uspořádaná trojice (SELinux identita, SELinux role, SELinux typ). 52, 53

SELinux politika Sada pravidel určujících řízení přístupu při nasazení SELinuxu. 52, 53

SELinux pravidlo Předpis specifikující množinu povolených operací, které mohou subjekty určitého typu provádět nad objekty určitého typu. 52, 53

SELinux pravidlo Předpis specifikující množinu povolených operací, které mohou subjekty určitého typu provádět nad objekty určitého typu. 53

SELinux role Množina operací, které smí subjekt konrétního typu v ní běžící vykonávat nad objekty. 51, 52, 218

SELinux typ Množina subjektů či objektů [Hol]. 52, 53, 218, 219

setgid (sgid) Příznak uložený v inode souboru mající vliv na EI procesu jej vykonávající. 41

setuid (suid) Příznak uložený v inode souboru mající vliv na EI procesu jej vykonávající. 40, 41

shell Program interpretující jemu předávané příkazy 61

shellcode Strojový kód vložený útočníkem do adresního prostoru procesu. 57, 59, 61, 62, 81, 217

slabika Jednotka množství informace. Dnes ve většině případů vyjadřuje to samé, co oktet – uspořádanou n-tici sestavenou z osmi bitů 21, 24

slovo Nejmenší jednotka, se kterou počítač pracuje, když zpracovává data. Délka slova v bitech je přímo úměrná maximální velikosti LAS 24

SMT Simultaneous Multithreading je technika vykonávání více vláken (mohou být původem z různých procesů) na jednom procesorovém jádru. Dnešní procesorová jádra běžně vykonávají instrukce mimo původní programové pořadí (out-of-order) a zároveň jich vykonávají více najednou, dovolí-li to okolnosti, čili nejsou-li dané instrukce na sobě závislé. V programech však zpravidla nebývá dostatečné množství nezávislých instrukcí k tomu, aby byl procesor plně využit. Proto byla vynalezena technologie SMT, kdy procesor přednačítá instrukce více vláken (v rámci architektury x86 typicky dvou) a redukuje tak čas čekání na načtení další informace z cache/paměti. 97, 212, 214

SOFTMODE Režim běhu, v němž jsou ochrany poskytované PaXem aktivovány pouze pro explicitně vyjmenované spustitelné soubory. 82

Sticky bit 41, 94

strojová instrukce Označení kódovaného příkazu pro provedení elementární operace procesoru, kterou je procesor schopen přímo vykonat 19

stránka Pojmenování jednotky mapování (bloku dat) mezi VAS a PAS umístěné v rámci VAS 26, 27, 67, 120, 216

stránkování Mechanismus virtualizace paměti. Více v sekci 24–26, 28

subjekt Aktivní entita provádějící operace nad objektem. V prostředí OS se jedná výhradně o proces. 15–17, 38, 39, 47, 51, 52, 81, 84–91, 199, 202, 203, 214, 217, 219

supervisor bit Jeden z příznaků vedených v každém záznamu tabulky stránek, jehož případné nastavení zamezí každému přístupu k dané paměťové stránce, pokud je veden při úrovni oprávnění nižší nežli Ring 0. 26, 76, 212

sysctl proměnná Proměnná v rámci Linuxu. Typicky modifikovatelná za běhu. Ovlivňuje chování jádra. 85, 100

tabulka stránek Datová struktura sloužící pro překlad VA na PA. Podrobněji v sekci 4.4.1.2 25–28, 64, 75, 76, 80, 212, 216, 218

tabulka systémových volání Klíčová datová struktura mapující čísla systémových volání na jim odpovídající funkce `sys_<jmeno>()`. 80

Task State Segment Segment stavu procesu je datová struktura uložená v paměti, která nese všechny informace o procesu. Každému procesu v systému je tato struktura přidělena. V případě přerušení jsou všechny informace o procesu potřebné pro jeho obnovení uloženy právě do TSS. 212, 218

TCSEC Dokument definující kritéria pro hodnocení bezpečnosti informačních systémů. 7, 9, 10, 15–18, 201–204, 212

TDI Dokument definující kritéria pro hodnocení bezpečnosti databázových aplikací 15, 212

text Mapování obsažené v AS procesu obsahující strojové instrukce programu. Více viz kapitola 4.6.1. 28, 29, 59, 70, 81, 216, 217

TNI Dokument definující kritéria pro hodnocení bezpečnosti počítačových sítí 15, 212

trampolína Posloupnost instrukcí generovaná jádrem v reálném čase. 78, 79, 210, 214

transakce Jakákoliv aktivita v systému vyvolaná subjektem s výjimkou autentizace subjektu. 47, 217

Translation Lookaside Buffer Vyrovnávací paměť umístěná v procesoru urychlující překlad VA na PA. Více v sekci 25–27, 75, 76, 210–212, 214

Trusted Computing Base Označení pro soubor všech částí informačního systému potřebných pro vynucení bezpečnostní politiky [Sir]. Částmi jsou hardwarové i softwarové mechanismy ochrany. 212

Trusted Path Execution Technika ochrany, která omezuje oprávnění ke spouštění souborů, které neleží v důvěryhodných umístěních. Typicky se jedná o skripty ležící v adresáři `/tmp`. 87, 212

Trusted System Důvěryhodný systém je systém, který zahrnuje jak hardwarová, tak i softwarová opatření umožňující jeho použití pro zpracovávání různě citlivých informací zároveň. [Pat] 212

Type Enforcement Řízení přístupu na základě příslušnosti subjektů a objektů k určitému SELinux typu. 212

UDEREF Technika, která zabraňuje neúmyslnému přístupu do User Space z jádra. Více viz kapitola 9.5.2. 10, 79, 80, 96

User mode Režim systému, v němž běží uživatelské procesy. Více v sekci 21, 24, 28, 63, 64, 74, 76, 123, 154, 212, 215, 219

User Space Adresní rozsah ve VAS přístupný strojovému kódu běžícímu v UM – uživatelskému procesu. Podrobněji popsáno v sekci 24, 29, 61, 63, 64, 67, 73, 74, 77, 79, 80, 83–85, 87, 94, 96, 123, 126, 140, 153, 200, 201, 212, 217, 219

userspace-stack Mapování obsažené v AS procesu obsahující dynamickou datovou strukturu nutnou pro fungování rekurzivního volání podprogramů. Více viz kapitola 4.6.7. 28–30, 58, 59, 61, 62, 73, 74, 78, 95, 96, 199, 217

uživatel Identita existující v rámci systému 51, 218

vanilla Přívlastek (užívaný hlavně ve spojení s jádrem) vyjadřuje, že se jedná o základní podobu entity, tak jak jí vytvořil její původní autor. V případě se tímto přívlastkem označuje jádro z hlavní vývojové větve. Většina distribucí naproti tomu obsahuje jádra upravená. Pojem má původ v základní příchuti zmrzliny v USA – vanilkové. Původně znamenal „bez dalších příchutí“ 13, 51, 54, 96, 98, 203

Virtual Address Space Adresní prostor paměti, který je uspořádán jinak nebo je dokonce větší, než je fyzicky připojená hlavní paměť. 20, 21, 23, 24, 26–29, 60–62, 67, 73, 76, 78, 81, 89, 203, 212, 213, 215, 217–219

x86 Instrukční sada používaná v IBM PC 10, 23, 26–28, 30–32, 34, 37, 62, 63, 75, 77, 80, 83, 84, 95, 125, 209, 212–214, 217–219

zranitelnost Slabé místo, které může být využito jednou nebo více hrozbami 7, 13, 31, 42, 57–59, 63, 67, 74, 94, 100, 203, 213, 214, 216, 217

závisle proměnná veličina Proměnná veličina, jejíž změny závisejí na dalších faktorech. Její hodnotu se snažíme nalézt. [Mec12] 113, 115, 118, 120, 137, 140, 142, 145, 154, 156, 159, 162, 164, 167, 169, 172, 174, 177, 180, 182, 213

čítač instrukcí Registr v procesoru, jehož účelem je adresovat instrukce strojového kódu v operační paměti počítače. Po načtení strojové instrukce z paměti – z adresy dané tímto registrem – se hodnota v něm uložená zvýší o velikost instrukce (včetně operandů) a stane se tak adresou následující instrukce. 211

čítač instrukcí Registr v procesoru, jehož účelem je adresovat instrukce strojového kódu v operační paměti počítače. Po načtení strojové instrukce z paměti – z adresy dané tímto registrem – se hodnota v něm uložená zvýší o velikost instrukce (včetně operandů) a stane se tak adresou následující instrukce 30, 211

řízení přístupu Mechanismus rozhodování tom, které subjekty mohou přistupovat k objektu. 38, 44, 51

A.13 Registry x86

EBP *Extended Base Pointer* je registr x86-kompatibilního procesoru obsahující adresu počátku aktivačního záznamu odpovídajícímu aktuálně vykonávané rozsahové jednotce. 29–35, 78

ECX *Extended Counter Register* je registr x86-kompatibilního procesoru používaný jako čítač pro počítadla cyklů. 78

EIP *Extended Instruction Pointer* je x86-kompatibilního procesoru obsahující adresu právě vykonávané instrukce. 29–31, 33–35, 59

ESP *Extended Stack Pointer* je registr x86-kompatibilního procesoru obsahující adresu vrcholu zásobníku. 29, 30, 32–35

A.14 Použité informační zdroje

[03] *Documentation for the PaX project*. 2003. URL: https://pax.grsecurity.net/docs/index.html.

[16] *Capabilities. overview of Linux capabilities*. 2016. URL: http://man7.org/linux/man-pages/man7/capabilities.7.html.

[17] *Pravidlo tří sigma*. 2017. URL: https://cs.wikipedia.org/wiki/Pravidlo_t%C3%85%C2%99%C3%83%C2%AD_sigma.

[85] *Trusted Computer System Evaluation Criteria*. 1. vyd. Washington: U.S. Department of defense, 1985. URL: http://csrc.nist.gov/publications/history/dod85.pdf.

[93] *Slovník výpočetní techniky*. Praha: Plus Publishing, 1993. ISBN: 8085297485.

[AddUti] *Grsecurity: Additional Utilities*. URL: https://en.wikibooks.org/wiki/Grsecurity/Additional_Utilities.

[AGcomp] *AppArmor and SELinux Comparison*. URL: http://www.suse.com/support/security/apparmor/features/selinux_comparison.html.

[Ale14] Aleatha. *Where are extended attributes stored?* 2014. URL: http://stackoverflow.com/questions/20618492/where-are-extended-attributes-stored.

[Ass] Tiago Assumpção. "Software Defense Mechanisms. PaX and the future". In: URL: http://www.cin.ufpe.br/~ruy/crypto/seguranca/Software_Defense_Mechanisms--PaX_and_the_future.ppt.

[Buc+08] Erik Buchanan et al. *Return-oriented Programming. Exploitation without Code Injection*. San Diego, 2008. URL: https://www.blackhat.com/presentations/bh-usa-08/Shacham/BH_US_08_Shacham_Return_Oriented_Programming.pdf.

[Buc15] Balazs Bucsay. "How to break out from various chroot solutions". 2015. URL: https://deepsec.net/docs/Slides/2015/Chw00t_How_To_Break%20Out_from_Various_Chroot_Solutions_-_Bucsay_Balazs.pdf.

[counter] *LinuxCounter*. 2011. URL: https://www.linuxcounter.net/statistics/kernel.

[CVE] *Linux Kernel Vulnerability Statistics. Vulnerability Trends Over Time*. URL: http://www.cvedetails.com/product/47/Linux-Linux-Kernel.html.

[CVE13] *CVE-2013-2141*. URL: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2013-2141.

[Čer09] Miroslav Čermák. *Řízení informačních rizik v praxi*. V Tribunu EU vyd. 1. Brno: Tribun EU, 2009. ISBN: 978-80-7399-731-1.

[Den] Peter J. Denning. "Fault Tolerant Operating Systems". In: *ACM Computing Surveys* vol. 8.issue 4 (), s. 359–389. ISSN: 03600300. DOI: 10.1145/356678.356680. URL: http://portal.acm.org/citation.cfm?doid=356678.356680.

[Dud04] Karel Dudáček. "Dekódování adres a návrh paměťového systému". In: (2004), s. 13.

[ElS] Hussein El-Sayed. "Linux Security Modules Framework". In: (). URL: http://se7so.blogspot.cz/2012/04/linux-security-modules-framework-lsm.html.

[GrSecDoc] *Grsecurity and PaX Configuration Options*. URL: https://en.wikibooks.org/wiki/Grsecurity/Appendix/Grsecurity_and_PaX_Configuration_Options.

[Háj15] Lukáš Hájek. *Security Enhanced Linux*. 2015. URL: http://kfe.fjfi.cvut.cz/~hajeklu2/files/AUX/Prezentace-SELinux/Hajek-AUX.pdf.

[Han93] *Hodnocení a klasifikace bezpečnosti informačních systémů*Petr Hanáček
Hodnocení a klasifikace bezpečnosti informačních systémů, 1993

. [HI12] Marco-Gisbert Hector a Ripoll Ismael. *Preventing brute force attacks against stack canary protection on networking servers*. 2012. URL: http://hmarco.org/data/Preventing_brute_force_attacks_against_stack_canary_protection_on_networking_servers.pdf.

[Hol] Martin Holas. *Povinné řízení přístupu*. Brno. URL: https://www.fi.muni.cz/~kas/p090/referaty/2005-podzim/st/selinux.html.

[Hor11] *Povinné řízení přístupu v Linuxu*Jan Horák
Povinné řízení přístupu v Linuxu, 2011

. [IntelArk] *Intel® Xeon® Processor E5-2620 v2*. 2013. URL: http://ark.intel.com/products/75789/Intel-Xeon-Processor-E5-2620-v2-15M-Cache-2_10-GHz.

[Kit+15] *Counteracting Data-Only Malware with Co de Pointer Examination*Thomas Kittel et al.
Counteracting Data-Only Malware with Co de Pointer Examination, 2015

. [Kor13] *Víceúrovňové systémy (MLS) v praxi*Jiří Kortus
Víceúrovňové systémy (MLS) v praxi, 2013

. [KPK12] *KGuard*Angelos Keromytis, Georgios Portokalidis a Vasileios Kemerlis
KGuard, 2012

. [KPK14] Vasileios Kemerlis, Michalis Polychronakis a Angelos Keromytis. *Ret2dir. Deconstructing Kernel Isolation*. Columbia University, 2014. URL: https://www.blackhat.com/docs/eu-14/materials/eu-14-Kemerlis-Ret2dir-Deconstructing-Kernel-Isolation.pdf.

[Krč17] Petr Krčmář. *Bezpečnostní záplaty GrSecurity už nebudou k dispozici zdarma*. 2017. URL: https://www.root.cz/clanky/bezpecnostni-zaplaty-grsecurity-uz-nebudou-k-dispozici-zdarma/.

[Kuč11] František Kučera. *Java a rozšířené atributy souborů*. 2011. URL: https://blog.frantovo.cz/c/320/Java%20a%C3%82%C2%A0roz%C3%85%C2%A1%C3%83%C2%AD%C3%85%C2%99en%C3%83%C2%A9%20atributy%20soubor%C3%85%C2%AF.

[Li] Kai Li. "Virtual Memory". In: URL: https://www.cs.princeton.edu/courses/archive/fall04/cos318/precepts/5.pdf.

[Luk13] Dejan Lukan. "Gentoo Hardening. Introduction to PaX and Grsecurity". 31.9.2013. URL: http://resources.infosecinstitute.com/gentoo-hardening-part-2-introduction-pax-grsecurity/.

[Luk14] Dejan Lukan. *Running VirtualBox/VMWare on Hardened Kernel*. 2014. URL: https://www.proteansec.com/linux/running-virtualboxvmware-hardened-gentoo/.

[Mec12] Erika Mechlová. *Role experimentu ve vědecké metodě*. Trojanovice, 2012. URL: http://pdf.osu.cz/dokumenty/projekt-cz1072300090024/role_experimentu_ve_vedecke_metode.pdf.

[MMC07] Frank Mayer, Karl MacMillan a David Caplan. *SELinux by example. using security enhanced Linux*. Upper Saddle River, NJ: Prentice Hall, c2007. ISBN: 9780131963696.

[Mošˇ] Petr Mošna. *Bezpečnostní model MAC v projektech GrSec a SELinux*. URL: http://download.zcu.cz/public/Prezentace/seminare%20CIV%202010/bezpecnost_linux/mosna-mac.pdf.

[Mül08] Tilo Müller. "ASLR Smack & Laugh Reference". In: *Seminar on Advanced Exploitation Techniques*. RWTH Aachen, Germany, 2008.

[NXexp] *NX bit explained*. URL: http://everything.explained.today/NX_bit/.

[Obe11] Jan Oberheide. *RELRO: RELocation Read-Only*. 2011. URL: http://blog.isis.poly.edu/exploitation%20mitigation%20techniques/exploitation%20techniques/2011/06/02/relro-relocation-read-only/.

[OberRos] Jon Oberheide a Dan Rosenberg. *Stack jacking. Your way to Grsec/Pax bypass*. URL: https://jon.oberheide.org/files/stackjacking-hes11.pdf.

[OKB17] Oleksii Oleksenko, Dmitrii Kuvaiskii a Pramod Bhatotia. "Intel MPX Explained. An Empirical Study of Intel MPX and Software-based Bounds Checking Approaches". In: (2017). URL: https://intel-mpx.github.io/.

[OSSec] "Operating Systems Security. Some basics". In: s. 11. URL: http://www.cse.chalmers.se/edu/year/2015/course/EDA263/oh15/L06%20OS%20Security.pdf.

[Pat] Wayne Patterson. "Inside the Orange Book". URL: http://www.howard.edu/csl/courses/cae-iae/lec-notes/sycs%20653/lecture_notes_12_fall_2010.ppt.

[Pel07] Tomáš Pelka. *Lokální útoky na operační systém linuxového typu*. 2007. URL: http://www.abclinuxu.cz/clanky/bezpecnost/lokalni-utoky-na-operacni-system-linuxoveho-typu.

[PKK14] *Ret2dir*Michalis Polychronakis, Angelos Keromytis a Vasileios Kemerlis
Ret2dir, 2014

. [Pol16] Polynomial. *How does reuse attack protection (RAP) work?* 2016. URL: http://security.stackexchange.com/questions/121942/how-does-reuse-attack-protection-rap-work/121944.

[RFC792] J. Postel. "INTERNET CONTROL MESSAGE PROTOCOL". 1981. URL: https://tools.ietf.org/html/rfc792.

[RFC793] "TRANSMISSION CONTROL PROTOCOL". Information Sciences Institute University of Southern California, 1981. URL: https://tools.ietf.org/html/rfc793.

[Ros10] Dan Rosenberg. *Restrict unprivileged access to kernel syslog*. 2010. URL: http://git.kernel.org/?p=linux/kernel/git/torvalds/linux-2.6.git;a=commitdiff;h=eaf06b241b091357e72b76863ba16e89610d31bd.

[Sir] Emin Gun Sirer. *Emin Gun Sirer*. URL: http://www.cs.cornell.edu/courses/cs414/2001sp/lectures/35-secmodels.pdf.

[SJ04] Peter Silberman a Richard Johnson. *A Comparison of Buffer Overflow Prevention Implementations and Weaknesses*. 2004. URL: https://www.blackhat.com/presentations/bh-usa-04/bh-us-04-silberman/bh-us-04-silberman-paper.pdf.

[SM94] Carl Staelin a Larry McVoy. *Lat_connect - measure interprocess connection latency via TCP/IP*. 1994. URL: http://manpages.ubuntu.com/manpages/zesty/en/man8/lat_connect.8.html.

[Spe03] Brad Spengler. "Grsecurity ACL documentation". 2003. URL: https://www.grsecurity.net/~spender/newDoc.pdf.

[Spe15] Brad Spengler. *Important Notice Regarding Public Availability of Stable Patches*. 26.8.2015. URL: https://grsecurity.net/announce.php.

[Spe16] Brad Spengler. *RAP is here. Public demo in 4.5 test patch and commercially available today!* 2016. URL: https://grsecurity.net/rap_announce.php.

[Spe17] Brad Spengler. *Passing the Baton*. 2017. URL: https://grsecurity.net/passing_the_baton.php.

[Sta] Richard Stallman. *Linux and the GNU System*. URL: http://www.gnu.org/gnu/linux-and-gnu.html.en.

[Sta02] Carl Staelin. *Lmbench3. measuring scalability*. 2002. URL: http://www.hpl.hp.com/techreports/2002/HPL-2002-313.pdf.

[Viv09] Gite Vivek. *Linux Kernel Security. SELinux vs AppArmor vs Grsecurity*. 2009. URL: http://www.cyberciti.biz/tips/selinux-vs-apparmor-vs-grsecurity.html.

[Zah11] Tomáš Zahradnický. *Bezpečnost a bezpečné programování. Přetečení bufferu*. Praha, 2011. URL: https://edux.fit.cvut.cz/oppa/MI-BPR/prednasky/bpr-3.pdf.