Západočeská Univerzita v Plzni
Fakulta Aplikovaných Věd
Katedra Informatiky a výpočetní techniky

Bakalářská práce

SID
SQL Injection Attack Detector

Plzeň, 2013

Jan Strnádek

Obsah

Poděkování

Rád bych touto cestou poděkoval Ing. Pavlu Královi, Ph.D. za odborné vedení a podnětné rady k bakalářské práci, dále pak panu Josefu Šimánkovi za odborné konzultace týkající se vývoje v Ruby a doporučení použitých knihoven.

Čestné prohlášení

Prohlašuji, že svou bakalářskou práci na téma „SID - SQL Injection Attack Detector“ jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborných zdrojů a literatury, které jsou citovány a uvedeny v seznamu literatury, popřípadě u některých zdrojů přímo v textu. Jako autor práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Plzni dne 7.5.2013 _____

Abstrakt

English

The main goal of this bachelor thesis is the analysis of the most recent attacks on web sites and web applications. In the theoretical part of this thesis chosen attacks have been thoroughly investigated and compared and the methods of prevention have been proposed. These methods cover wide range of possible attacts both with and without frameworks.

The practical part has been focused on development of the penetration testing tool for detection of one of the most dangerous security risk - SQL Injection. The suggested tool uses parameters that are automatically changeable to be fitted for the certain web site or application. The changes in the web site or web application are recorded and compared in order to detect potential security issues.

The tool have been evaluated with existing tools. Unfortunately, there has been found no similar tool available for free with a direct detection function for SQL Injection. Therefore the most similar freeware tools trying to test this security threat have been used for the comparison.

Česky

Hlavním cílem této bakalářské práce je analýza nejčastějších typů útoků na webové stránky a aplikace. V teoretické části jsou důkladně analyzovány vybrané útoky a jsou ukázány metody prevence s i bez použití frameworků.

Praktická část byla zaměřena na vývoj nástroje pro penetrační testování určeného k detekci jednoho z nejnebezpečnějších bezpečnostních rizik - SQL injection. Navržený nástroj využívá změny parametrů a vyhodnocení změn obsahu na stránce před a po úpravě.

Nástroj byl porovnán s existujícími nástroji a byly analyzovány výsledky. Bohužel zde bylo zjištěno, že neexistuje volně dostupný nástroj, který by nabízel shodnou funkcionalitu. Proto byly v testu použity nástroje požadovaných vlastností.

Úvod

V dnešní době je internet synonymem pro používání počítače, tabletu, smartphone a jiných zařízení. S tím rozhodně souvisí otázka bezpečnosti uživatelských dat. Většina uživatelů bohužel využívá všude stejné heslo, proto obezřetnému útočníkovi stačí získat login a heslo z jedné databáze a zkusit to i jinde. K této situace došlo nedávno při napadení porno stránek (*pron.com*) skupinou „LulzSec“[1], která následně získaná data zveřejnila[6]. Hesla nebyla v databázích nijak „hashována“[2], byla uložena v podobě otevřeného textu, a proto nebyl problém vyzkoušet se přihlásit do emailových schránek, popřípadě dalších jiných webových služeb, které tito uživatelé využívali. Nikdy přesně nevíme, komu vlastně data svěřujeme a jaké bezpečnostní opatření je dotyčnou firmou či osobou zajištěno! Data jsou uchovávána v mnoha databázových systémech a jednou z hlavních otázek je také bezpečnost těchto dat. Možností útoků na webové aplikace, webové stránky nebo přímo servery je mnoho. Významnější budou popsány v následující kapitole.

Cílem této práce je seznámit čtenáře s bezpečnostními riziky webových aplikací, metodami prevence rizik a dále podrobnější seznámení s problémem SQL injection. Cílem praktické části bude vytvoření nástroje pro penetrační testy určeného k detekci těchto bezpečnostních rizik.

Celá tato práce je především zaměřena na detekci problému SQL injection. Aktuálně je podle serveru *http://techworld.com* za zhruba 97% úniků dat zneužití právě této chyby.

[1] Lulz Security - tato skupina stála i za útokem na Sony Pictures v roce 2011, kde právě díky SQL Injection odcizila velké množství dat.

[2] Hash - algoritmus pro převedení vstupních dat do unikátního otisku, tato funkce „by měla být jednosměrná!“

Kapitola 1

Útoky na webové aplikace

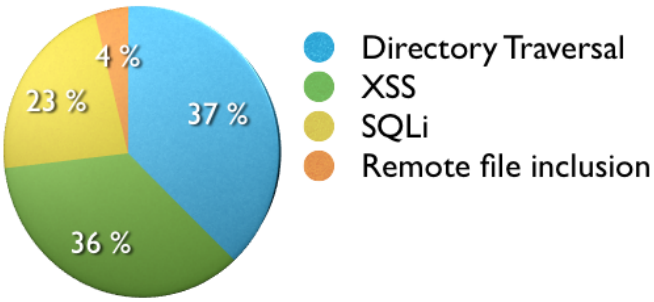
1.1 Přehled

Přehled typů útoků, které budou dále detailně rozebrány.

- XSS - Cross-Site scripting
  - Využití nechráněných vstupů pro vložení vlastního JavaScriptového kódu.
- DT - Directory traversal
  - Získávání zdrojových souborů přes špatně nastavené direktivy.
- CSRF - Cross-Site Request Forgery
  - Volání nelegitimních akcí z legitimního zdroje.
- PHP remote upload and execution scripts
  - Zneužití nahrávaných souborů přes webové formuláře.
- SQL Injection - normal / blind
  - Úpravy databázových dotazů přes neochráněné vstupy.
- a další, protože webový server je jen počítač s operačním systémem, který obsahuje bezpečnostní chyby

1.2 Nejčastější typy útoků

Se zajímavými daty přichází server *http://cnet.com*, který uvádí, že každé 2 minuty je napadnuta nějaká webová stránka. Zastoupení typů provedených útoků je vidět na následujícím grafu 1.1.



Obrázek 1.1: Graf nejčastějších typů útoků [Zdroj: *http://cnet.com*]

1.3 XSS - Cross-Site scripting

XSS využívá podobně jako SQLi neochráněných vstupních proměnných na webových stránkách. Díky nim může do aplikací podstrčit svůj vlastní (například JavaScriptový) kód, což může následně využít k získání dat (zejména cookies od uživatelů), zastavení webových stránek atd. Existují dva základní typy XSS útoku:

1.3.1 Nepersistentní

Tento typ využívá nezabezpečených vstupních proměnných z URL adresy / POST dat[1], které jsou vypisovány na stránku. Útočníkovi stačí URL upravit a nějakým způsobem (například sociálním inženýrstvím, podvrženým emailem z banky apod.) donutit uživatele na tento odkaz kliknout.

1.3.2 Persistentní

Tento typ je mnohem nebezpečnější, protože na napadené stránky se nevstupuje přes upravenou URL adresu, ale kód se vykonává automaticky. Tato chyba se často objevuje v různých diskusních fórech či návštěvních knihách, kde se nevalidují vstupy. Do těchto nezabezpečených vstupů stačí útočníkovi pouze vložit Java Scriptový kód, který se následně provede každému, kdo tuto stránku otevře.

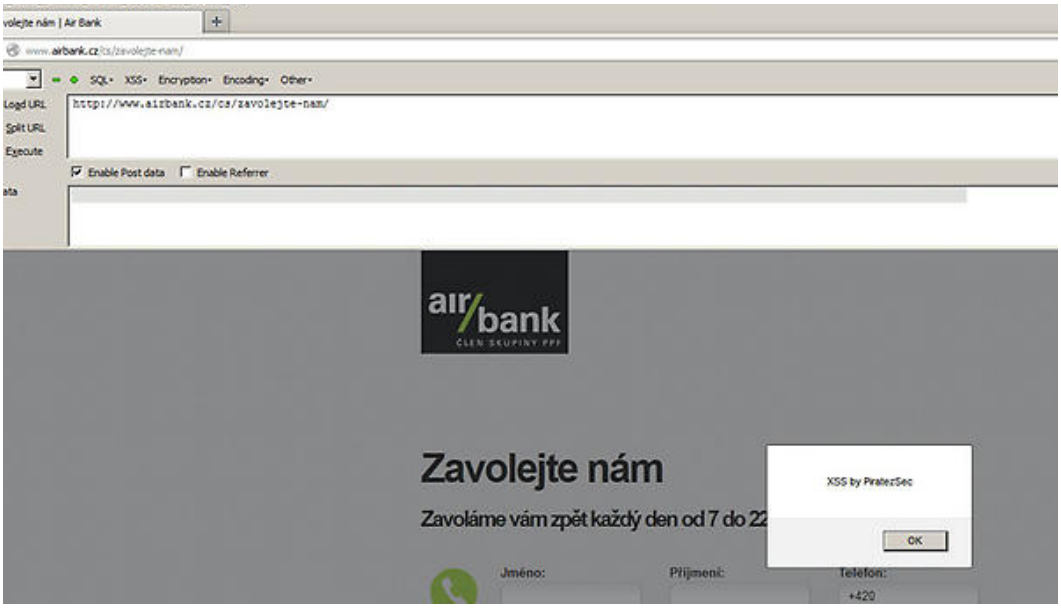
[1]GET a POST jsou základní způsoby jak přenést parametry na další stranu, rozdíl je v tom, že GET jsou vidět v URL a POST ne. Ale oba dva lze bez problémů podvrhovat!

1.3.3 Ukázka

Na obrázku 1.2 můžeme vidět úspěšný nepersistentní XSS útok na serveru air/bank. (Bohužel nebyly zveřejněny detaily a byl zveřejněn pouze tento obrázek v dost špatné kvalitě.) V nechráněném vstupu byl zadán kód výstražné hlášky javascriptu:

Kód 1.1: Výstražná hláška v jazyce JavaScript

```
<script type="text/javascritpt">alert("XSS by PiratezSec");</script>
```



Obrázek 1.2: Non-persistent XSS [Zdroj: https://twitter.com/Czechurity]

Tato chyba byla objevena skupinou Czechurity (*https://twitter.com/Czechurity*), která má na svědomí i kompromitaci webových stránek Unicredit bank v březnu 2013, které bylo médii chybně interpretováno jako DDoS[2].

1.3.4 Různé varianty zapsání XSS

Výstupy na webových stránkách mohou být různým způsobem filtrovány, popřípadě může být přímo filtrován script tag. Toto lze bohužel jednoduše obejít[2]. Zde několik příkladů (ale ne všechny prohlížeče toto interpretují):

Kód 1.2: Schování JavaScriptu do neexistujícího obrázku

[2]Distributed Denial of Service - útok, který zahlcuje službu, až dojde k jejímu pádu nebo nedostupnosti pro ostatní uživatele.

```
<img src="javascript:alert('XSS');">
```

Kód 1.3: Zakázané uvozovky? Nahrazení entitami ...

```
<img src=javascript:alert(\&quot;XSS\&quot;)>
```

Kód 1.4: Další možností je převedení na unikód

```
<img src=&#106;&#97;&#118;&#97;&#115;&#99;&#114;&#105;
&#112;&#116;&#58;&#97;&#108;&#101;&#114;&#116;&#40;
&#39;&#88;&#83;&#83;&#39;&#41;>
```

1.3.5 Obrana

Obrana před XSS není snadná. Velké webové portály mají desítky různých vstupů, které je třeba hlídat. Pokud se rozhodneme využít nějaký framework, pak je ideální obranou výběr takového, který právě těmto incidentům dovede předcházet (Nette Framework[3], Ruby on Rails[4] apod.). Dále je kvůli zamezení persistentním útokům třeba striktně hlídat, co ukládáme do databází či do jiných úložišť.

1.3.6 Důsledky

Asi nejzásadnějším důsledkem úspěšného XSS útoku bude tzv. *sessions hijacking*. Je to odcizení sessions_id (sessions_key), což je unikátní ID uživatele pro přihlašování ke vzdáleným serverům. Toto ID je možné zjistit několika způsoby, ale nejúčinějším způsobem, jak jej zjistit, je právě pomocí XSS. Pokud útočník toto ID získá, může se autorizovat k zabezpečenému serveru.

1.4 Directory traversal

Webový server slouží hlavně pro poskytování souborů, které mohou být statické (obrázky, css styly, HTML soubory) nebo dynamické (Ruby, PHP, ASP atd.). Pokud vytvoříme požadavek na web server, pak nám server při statickém obsahu soubor okamžitě interpretuje, ale při dynamickém ho nejdříve zpracuje a pak interpretuje. Při útoku typu „directory traversal“ (někdy se mu taky říká „path traversal“)[4] využívá útočník špatného či dokonce žádného omezení přístupu k souborům právě pomocí dynamického obsahu webového serveru.

[3] http://nette.org
[4] http://rubyonrails.org

1.4.1 Příklad directory traversal útoku

Mějme:

Kód 1.5: URL adresa s podezdřením na include souboru

```
http://portal.czu.cz/index.php?item=novinky.html
```

Při bližším zkoumání vidíme, že se zde pravděpodobně vkládá soubor „novinky.html“ do stránky, a to odněkud ze souborového systému. Otázkou však zůstává, co se bude dít, budeme-li tento parametr měnit ručně, a kam až se dostaneme.

Kód 1.6: Manipulace s URL - získání .htaccess

```
http://portal.czu.cz/index.php?item=../.htaccess
```

Kód 1.7: Manipulace s URL - získání config.neon

```
http://portal.czu.cz/index.php?item=../config/config.neon
```

V prvním příkladě jsme se snažili získat soubor „.htaccess“, který může obsahovat autentizační metody, nastavení práv apod. Ve druhém příkladě jsme se pokoušeli získat soubor „config.neon“. Tento soubor slouží pro uchování citlivých informací na webech, které využívají Nette Framework. Příkladem takových informací může být přihlášení k databázi.

1.4.2 IIS Web Server

Starší verze IIS[5] umožňovaly dokonce i vykonávat příkazy na serveru.

Kód 1.8: Ukázka URL pro „děravé“ IIS

```
http://iis.czu.cz/scr/..%5c../winnt/system32/ cmd.exe?/c+dir+c:\
```

Tento příkaz spustil „cmd.exe“ (příkazová řádka systému Windows) a v něm příkaz „dir c:\“. Nic nám tedy nebrání přidávat uživatele do systému, nebo formátovat pevné disky, jak lze vidět zde:

Kód 1.9: Formátování disku C: přes chybu v IIS

```
http://iis.czu.cz/scr/..%5c../winnt/system32/ cmd.exe?/c+format+c:\
```

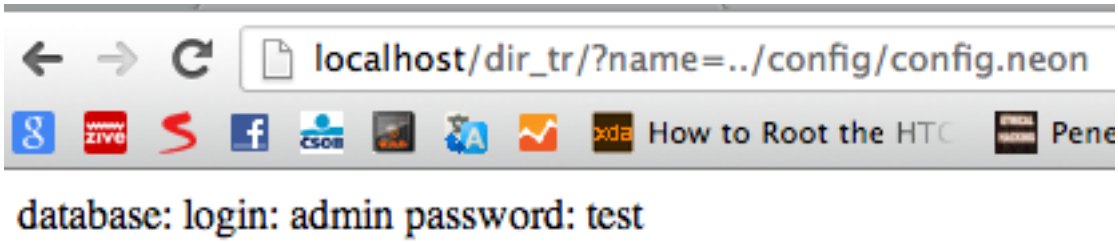
[5]Internet Information Service - Microsoftem distribuovaná obdoba webového serveru Apache2

1.4.3 Obrana

- Mít správně nastavená oprávnění a cesty jednotlivých webových serverů (virtuálních hostů).
- Kontrolovat, co vlastně do stránky vkládáme.
- Úplně se vyhnout vkládání dat do stránky (například v PHP můžeme použít *spl_autoload_register*, který nám automaticky podle zadané funkce načítá třídy ze souborového systému).

1.4.4 Praktická ukázka

Na obrázku 1.3 vidíme úspěšný directory traversal útok na vkládání v parametru „name“. Ze získaného souboru „config.neon“ se následně dozvíme, že přihlašovací jméno k databázi je „admin“ a heslo je také „test“.



Obrázek 1.3: Directory traversal - zobrazení obsahu souboru config.neon

1.5 CSRF - Cross-Site Request Forgery

U tohoto typu útoku většinou potřebujeme „osobu uvnitř“, která má dostatečná oprávnění a my jsme schopni jí přesvědčit (často pomocí sociálního inženýrství), aby spustila nebo otevřela námi upravenou URL. Tento útok využívá situace, že sice přijde požadavek na vykonání určité akce od legitimního uživatele, ale na nelegitimní zdroj.[3] (Tento postup často vyžaduje znalost URL pro různé akce na webové stránce.)

1.5.1 Příklad CSRF

Jednoduchým příkladem může být jakýkoliv redakční systém. Nejjednoduší je, pokud server, na který útočíme, používá nějaký známý CMS[6] - napří-

[6]Content Managment System - systém zajišťující správu webového obsahu

klad Joomla[7], Drupal[8] a další. Zde URL adresy pro vykonávání určitých akcí známe, protože si je můžeme vyzkoušet sami. Mějme tedy redakční systém, který má script *admin.php* a například tyto parametry:

- **action** - Která akce bude provedena
- **user** - Uživatel
- **hodnota** - Nějaká další hodnota

Můžeme tedy spustit například toto:

Kód 1.10: URL změny uživatelské role

```
http://portal.czu.cz/admin.php?action=changeRole&user=2&role=admin
```

Jestliže tento příkaz (*změna role uživatele „2“, což jsme například právě my, na roli „hlavního administrátora“*) zavoláme jako neautorizovaná osoba, příkaz se neprovede a bude nám vypsáno, že nemáme dostatečná oprávnění. Jestliže ovšem zašleme podvodný email správci portálu, který na tento link klikne a bude zároveň přihlášen na zmíněné stránce http://portal.czu.cz, tento příkaz proběhne bez problémů, uživatel Honza bude mít práva „hlavního administrátora“, což už je značný bezpečnostní problém.

1.5.2 Obrana proti CSRF

Nejúčinější obranou proti CSRF je generování a kontrolování tokenů. Do každého formuláře, popřípadě i odkazu, přidáme tzv. „token“ (tj. náhodně vygenerovaný řetězec, příklad URL s tokenem viz 1.11), který se uloží a následně přidá do každého formuláře / odkazu na aktuální stránce. Při přechodu na další stránku se přijatý token ověří proti uloženému. Pokud je vše v pořádku, akce se provede. Pokud token nesouhlasí, je uživatel přesměrován na „bezpečnou“ stránku, která jej informuje o neplatné akci. Tato metoda obrany je založena na tom, že útočník není schopen token předvídat. Samozřejmě, pokud by byla chyba při generování tokenu, popřípadě by se z nějakého důvodu neměnil, je zde možnost, že jej útočník zjistí.

Kód 1.11: CSRF obrana - token

```
http://czu.cz/admin.php?action=changeRole&user=2&role=1&token=ad70CZf82
```

[7] http://www.joomla.org/ - PHP CMS
[8] http://www.drupal.org/ - Open CMS system

1.6 Remote execution script

Tento typ útoku využívá situaci, kdy můžeme pomocí formuláře pro nahrání souborů nahrát PHP skript, který je dostupný pomocí URL a je web serverem vykonáván. Správně vytvořený PHP skript pak může naše akce směrovat pomocí příkazů (system() a eval()) na konzoli stroje a následně nám umožňuje další činnost. Jednou z možností je využití scriptu pro vytvoření reverzního shellu[9]. Tento skript umí vytvořit například oblíbený Metasploit framework[10], který pak skýtá opravdu široké využití.

1.6.1 Obrana

Zde je obrana jednoduchá, a to dávat si pozor na to, co je nahráváno. Soubory můžeme tedy jednoduše přejmenovávat (změnit příponu z interpretovaných - *.php*, *.php3* například na *.txt*, které není interpretováno) nebo zakázat jejich vykonávání. Příklade Další užitečnou funkcí pro „předejití“ problémům je zákaz funkcí *eval()* a *system()*. Tyto funkce umožňují volání systémových funkcí (spouštění programů, kopírování souborů atd.). Značný problém je, když webový server běží pod účtem superuživatele (*root*). Útočník tedy získává automaticky práva hlavního administrátora, což může mít neblahé následky. Proto se webové servery a další služby spouští pod speciálními uživateli s omezenými právy.

1.7 Open Directory browsing

Další ukázka špatně nastaveného serveru. Jsme totiž schopni zjistit adresářovou strukturu projektů a z ní vyčíst mnoho informací, které měly zůstat skryty. Příkladem mohly být dříve používané soubory s příponou *.inc*, které bylo možné číst, protože je PHP interpret standardně nevykonával. Tyto soubory je stále možné nalézt pomocí internetových vyhledávačů (často byly vyhledávány soubory s názvem *config.php.inc*, které obsahovaly většinou údaje pro připojení k databázi a jiné konfigurační údaje).

[9] Reverse Shell - otevření spojení z cílového stroje na náš počítač „jakoby“ SSH obráceně

[10] Metasploit framework je velice oblíbený penetrační tester, který lze získat zdarma na: http://www.metasploit.com/

1.8 SQL Injection

Podobně jako XSS využívá SQL injection neochráněné vstupy, ovšem využívá je jako útok na databázovou vrstvu, protože pomocí neochráněných vstupů upravuje SQL dotazy volané nad databázovým serverem. (Více o SQL injection bude rozebráno v kapitole 3)

1.9 Frameworky

Dnes jsou v oblibě frameworky pro rychlejší a snažší vývoj webů. Některé z nich kladou důraz na obranu proti XSS, CSRF i SQLi, ale lze jim věřit? U většiny frameworků je zmíněno tzv. „escapování“, což znamená převedení „nebezpečných znaků“: „<>¨“ atd. na „bezpečné“, které nejsou prohlížečem nebo SQL interpretem vykonávány. Jestliže vypisujeme neescapovanou proměnnou, může to znamenat bezpečnostní riziko! Dále budou uvedeny základní bezpečnostní nedostatky některých z frameworků, které byly objeveny za posledních několik měsíců.

1. *Zend Framework - PHP*
   - *http://framework.zend.com/*
   - *SQL Injection a XSS* - neprovádí escapování a je nutné využít další funkce (stejně jako v čistém PHP)
   - *CSRF* - obrana proti CSRF je plně na bedrech programátora
2. *Ruby on Rails (RoR) - Ruby*
   - *http://rubyonrails.org/*
   - *SQL Injection* - 2. 1. 2013 byla objevena zásadní chyba tohoto typu v modulu Active Record, který Ruby on Rails využívá jako ORM[11]. Chybou, která byla označena jako CVE-2012-5664, jsou postiženy všechny verze Ruby on Rails: *https://groups.google.com/forum/#!topic/ rubyonrails-security/DCNTNp_qjFM*. Veškeré dotazy jsou striktně escapovány, ale pro určité položky to lze vypnout tzn. položky, u kterých bylo escapování vypnuto se nebude nic ověřovat a tyto položky mohou být bezpečnostním rizikem.
   - *CSRF* - Tokeny jsou přidávány ke všem formulářům automaticky od Ruby on Rails verze 2.

[11] ORM - objektové relační mapování - mapuje data z databází na objekty

- *XSS*
  - RoR ve verzi 2 nehlídala výstupy v šablonách a bylo nutné používat helper[12].
  - RoR ve verzi 3 hlídá vše, ale je možné vynutit vypsání normální (hodí se například pokud máme „před tím“ postaven Markdown[13], který vstup a výstup hlídá sám).

3. *DJango - Python*
   - *https://www.djangoproject.com/*
   - *SQL Injection*
     - Querysets - ORM - hlídají proměnné automaticky, lze vynutit, aby se tak nedělo
     - RAW queries - neescapují vůbec
   - *CSRF* - Django obsahuje *middleware*[14], který nám umožní přidávat CSRF token k formulářům a následně ho ověřovat: - více informací na: *https://docs.djangoproject.com/en/dev/ref/contrib/csrf/*.
   - *XSS* - Šablony DJanga automaticky escapují proměnné, ale ne všechny (více informací na: *https://docs.djangoproject.com/en/ dev/topics/security/*).
4. *Nette Framework - PHP*
   - *http://nette.org/cs/*
   - *SQL Injection* - při využití *Nette\Database*[15] se escapují všechny proměnné automaticky (ovšem lze zde vynutit neescapování).
   - *CSRF* - Nette\Forms[16] mají volbu zapnout přidávání CSRF tokenu a následně ho sami ověřují (pomocí metody *addProtection()*).
   - *XSS* - Veškeré proměnné vypisované do šablon jsou automaticky escapovány. Lze opět vynutit, aby k tomu nedocházelo.
5. *ASP - C#*
   - *SQL Injection* - Pokud je použita výchozí databázová vrstva ASP, escapuje se úplně vše a není možné tuto funkčnost vypnout. Pokud je ovšem SQL dotaz napsán ručně, je to čistě na programátorovi.

[12] Jednoduché makro používané v šabloně.
[13] Nástroj pro převádění textu do HTML pomocí speciálních značek.
[14] Middleware je tzv. „prostředník“, v tomto případě mezi jádrem DJanga a naší aplikací.
[15] *Nette\Database* je název vrstvy pro jednodušší práci s databází
[16] Třída reprezentující formuláře v Nette

- *XSS* - Striktně se escapují všechny proměnné, které se vkládají do šablon.
- *CSRF* - Tokeny ve formulářích nejsou implicitně zapnuty, ale je možné využít *ViewStateUserKey*, popřípadě knihovny třetích stran.

Kapitola 2

SQL Injection (SQLi)

2.1 Popis

SQL Injection podobně jako XSS využívá neochráněných vstupů, ovšem cílem je napadání databázové vrstvy (viz dříve). Pomocí neochráněných vstupů jsme schopni upravovat SQL dotazy, vkládat do nich podmínky, popřípadě vnořené dotazy.

2.2 Příklad SQL Injection

Mějme tabulku (například v databázovém systému *MySQL*) se seznamem písniček. Ukázku tabulky můžeme vidět v tabulce:

id	category_id	autor	name
1.	**2**	Celldweller	One good reason
2.	**3**	Asonance	Království Keltů
3.	**2**	Celldweller	EON
4.	**4**	Hectix	Return
5.	**3**	Blue Stahli	Takedown

Tabulka 2.1: Tabulka hudebního katalogu

Ve webové aplikaci přejdeme na URL:

Kód 2.1: URL webové aplikace

```
http://localhost/songs.php?categoryId=2
```

Script *songs.php* načte parametr *categoryId* a podle něj vytvoří dotaz, který vybere písničky z dané kategorie:

Kód 2.2: Vytvořený SQL dotaz

```
SELECT * FROM songs WHERE category_id = 2
```

Dotaz bude vykonán a na webové stránce se zobrazí pouze písničky z kategorie číslo 2. Pokud bychom ale URL ručně přepsali a nahradili bychom kritickou část, například:

Kód 2.3: Ručně upravené URL

```
http://localhost/songs.php?categoryId = 2 OR 1 = 1
```

a script by nebyl ochráněn proti těmto „nevhodným“ vstupům, zachoval by se stejně a vygeneroval by následující dotaz:

Kód 2.4: Vygenerovaný SQL dotaz z upraveného URL

```
SELECT * FROM songs WHERE category_id = 2 OR 1 = 1
```

Tento dotaz je ovšem úplně jiný, vrací totiž všechny skladby.

2.3 Rozdíl mezi SQL Injection a Blind SQL Injection

Podstata útoku je v obou případech stejná, ovšem u *Blind SQL Injection* nevidíme výsledek, což znamená delší hledání problému. Oproti tomu v předchozím případě jsme výsledek viděli ihned (zobrazily se všechny skladby a ne pouze daná kategorie), což znamenalo odhalení tohoto problému.

2.4 Předcházení a obrana

- Kontrola příchozích dat na aplikační vrstvě - pokud vím, že mi v parametru *category_id* má přijít číslo, tak budu validovat číslo.
- Využití funkcí pro „přepsání“ speciálních znaků do entit (v php např.: *mysql_real_escape_string* ...). Tyto funkce nahradí znaky, které by mohly SQL dotaz nějakým způsobem *upravit* nebo *poškodit* na text.
- Využití databázové vrstvy, která má jako jeden z cílů právě předcházení těmto rizikům (příkladem může být *Dibi - Database Abstraction Library pro PHP*[1]).
- Správně nastavená oprávnění - pro připojení webových aplikací využívat speciálního uživatele s omezenými právy (pokud je z nějakého důvodu nepotřebujeme). Ideálně aplikace a k ní konkrétní uživatel s jednou databází, který nikam jinam nemůže (zabránění, aby se z DB jedné

[1] Je zdarma k dispozici na http://dibiphp.com

aplikace dostal i do dalších), dále zákaz nepotřebných příkazů pro tyto uživatele (například exec, drop, alter). Pak i kdyby útočník objevil SQL Injection chybu, tak nám například nemůže vymazat všechny tabulky.

Kapitola 3

Důsledky

Důsledky napadení nedostatečně zabezpečené aplikace mohou být fatální, ať je to získání administrátorského přístupu do webové aplikace, odcizení dat nebo až získání úplné kontroly nad cílovým serverem.

3.1 Ukázky možného napadení

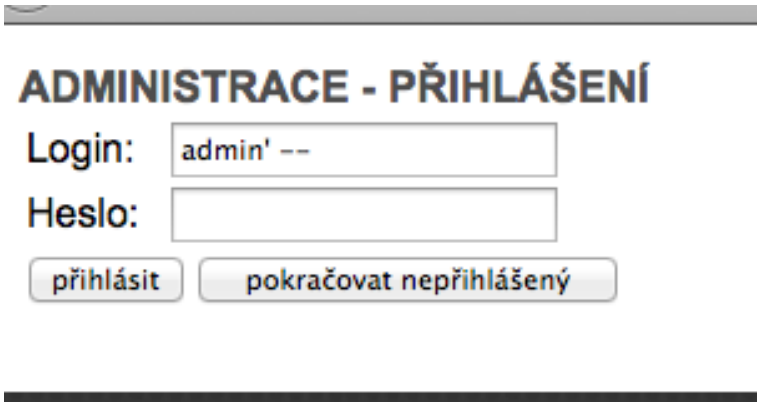
3.1.1 SQLi v redakčním systému

Tato chyba byla nahlášena na *http://exploit-db.org/* a její kód zveřejněn. Byly zasaženy desítky webů používajících redakční systém WordPress. Z obrázku je vidět, že parametr *id* není bezpečný, tudíž je možné dotaz upravit a získat například uživatelská jména či hesla.

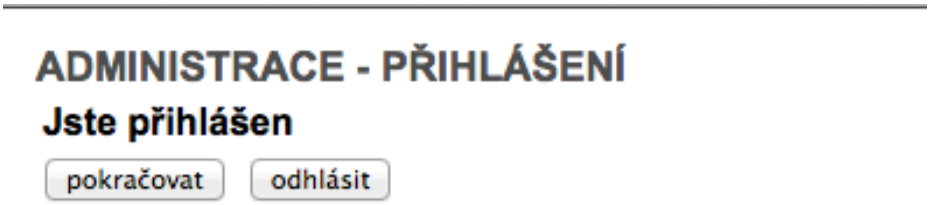
Obrázek 3.1: SQL injection v redakčním systému

3.1.2 Chybný přihlašovací formulář

Ukázka chybně zabezpečného vstup přihlašovacího formuláře na nejmenovaném portálu české firmy, kde je možné se pomocí zakomentování zbytku SQL dotazu (ověření hesla) přihlásit jako administrátor.



Obrázek 3.2: Zápis SQLi



Obrázek 3.3: Úspěšné přihlášení

3.1.3 Sony Pictures - 2011

Proti společnosti Sony byly v roce 2011 několikrát použity DDoS útoky a nakonec vše vyústilo v odcizení dat skupinou LulzSec[1] pomocí SQL injection. I taková velká společnost jako Sony, měla údaje v databázích v nešifrované podobě. Hackerům se podařilo odcizit 1 milion uživatelských dat (jména, hesla, adresy a datum narození), dále hesla administrátorů a mnoho dalšího - více v *http://www.thewhir.com/web-hosting-news/hackers-attack-sony-pictures-with-single-sql-injection*. Většina členů této skupiny byla nedávno zatčena a pochytána, díky chybě hlavního člena s přezdívkou „Sabu“ a jeho následné spolupráci s FBI zdroj *The Guardian*[12].

You call it war, we laugh at your battleships.
(Heslo skupiny LulzSec)

[1]Lulz Security - skupina počítačových hackerů, jejich webové stránky byly zablokovány, ale twitter zůstal *https://twitter.com/LulzSec*

Kapitola 4

Návrh nástroje pro penetrační testování

Pro detekování chyb typu SQL Injection je v této části vysvětlen postup a princip fungování nástroje pro penetrační testování. V další části 5.1 je uveden detailně postup od zpracování webové stránky až po metody testování a rozpoznání SQL Injection chyby.

4.1 Postup testu

1. Analýza zdrojového kódu
    - Získání objektové reprezentace webové stránky použitelné pro další zpracování.
2. Vytvoření stromu webové stránky
    - Extrahování formulářů a hypertextových odkazů z webové stránky (pro vytvoření stromu).
    - Normalizace URL adres kvůli detekování přechodu na jiné domény a sjednocení všech možných kombinací relativních a absolutních URL adres.
3. Otestování hypertextových odkazů
    - Otestování možnosti SQLi v hypertextových odkazech pomocí analýzy webové stránky (změna stránky - počtu elementů v závislosti na úpravě sql dotazu).
4. Otestování formulářů
    - Stejný postup jako u hypertextových odkazů.
5. Reprezentace výsledků
    - Reprezentace výsledků pro další zpracování - na standardní výstup, popřípadě ve specifickém formátu do souboru.

4.2 Analýza zdrojového kódu stránky

Při analýze stojí za úvahu, zda si napsat vlastní HTML parser nebo použít nějaký stávající, čímž by bylo rozhodnuto i o jazyce, ve kterém bude testovací skript napsán. Bylo možné použít následující HTML parsery:

- XML Reader - PHP
- Simple HTML DOM - PHP
- Nokogiri - Ruby
- Hpricot - Ruby
- JSoup - Java

4.2.1 Testování parserů

Pro testování jsem si napsal 2 webové stránky, první s validním a správně napsaným HTML a druhou s rozházenými tagy, nedokončenými uvozovkami, která by měla více reprezentovat reálný model webových stránek. I když existují xHTML standardy je většina projektů psaná „ve spěchu“ a kód bývá dost často nepořádný. Proto je nutné tento problémem zohlednit.

XML Reader - PHP - Tento parser je primárně pro parsování validních XML dokumentů, proto se ukázal pro parsování HTML stránek jako nevhodný.
(Ke stažení na: http://php.net/manual/en/book.xmlreader.php)

Simple HTML DOM - PHP - Parser v PHP určený přímo pro parsování HTML DOM Modelu se jevil jako vhodný kandidát, ale při otestování na reálném webu (sestavování stromu webové stránky) se parser ukázal jako nepoužitelný. Velmi špatně reagoval na další chyby v HTML stránce a nebylo možné je rozumně zachytávat. Také jeho výběr prvků a následné parsování dat bylo nutné kombinovat s regulárními výrazy. Ze zmíněných důvodů jsem od tohoto parseru upustil.
(Ke stažení na: http://simplehtmldom.sourceforge.net/)

Nokogiri - Ruby - První parser v testu, který si bez problémů poradil s parsováním špatných HTML stránek, byl schopen rozpoznat správně odkazy, chybné uzavírání parametrů. Další obrovským přínosem je, že funguje na principu CSS3 selektorů:

Kód 4.1: CSS3 Selektory NokoGiri

```
# Load Nokogiri::HTML Document for page
doc = Nokogiri::HTML(open('http://www.google.com/search?q=sparklemotion'))

# Search for nodes by css
doc.css("body a").each do |link|
        puts link.content
end
```

Dále implementuje XPath a elementy dále rozděluje (ne jako předchozí parser Simple HTML DOM, který vrací pouze textovou podobu).

Kód 4.2: Parsování

```
# Search for nodes by css
doc.css("body a").each do |link|
        puts link['href'] # Print href attribute
end
```

(Ke stažení na: http://nokogiri.org/)

HPricot - Ruby - Další parser v Ruby, který je hodně podobný předchozímu Nokogiri. Opět si poradí s CSS3 selektory, Xpath výrazy atd. Má jedinou nevýhodu: není již dále vyvíjen a podporován.
(Ke stažení na: https://github.com/hpricot/hpricot/)

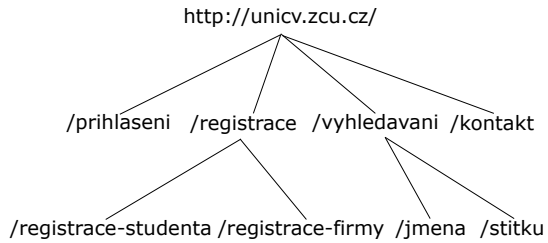
JSoup - Java - První HTML parser, který je napsán v programovacím jazyce Java. Podporuje XPath, ale pouze CSS selektory. Největší nevýhodou je samotný programovací jazyk. Vývoj v Javě je zdlouhavý, a proto jsem volil rychlejší programovací jazyky, kterým je Ruby a PHP. Další nevýhodou Javy je, že není možná integrace do Metasploit frameworku (o Metasploitu viz kapitola 2.6), který je napsán v Ruby.
(Ke stažení na: http://jsoup.org/)

4.2.2 Závěr testu parserů

Každý z testovaných parserů měl své výhody a nevýhody, nicméně z testů vyšel jednoznačně nejlépe parser *Nokogiri*, který určil i programovací jazyk, ve kterém penetrační test bude napsán - *Ruby ve verzi 1.9.3*.

4.3 Vytváření stromu webových stránek

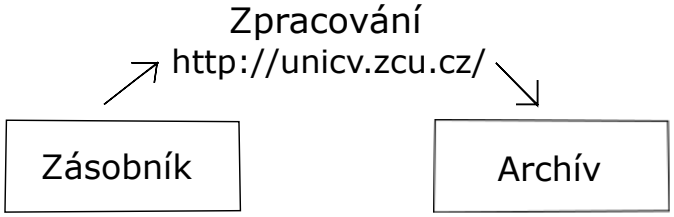
Abychom mohli získat všechny odkazy a formuláře na webových stránkách, musíme určit URL strom (viz obrázku 4.1) a z něj následně získat všechny odkazy a formuláře. Získávání stromu je postaveno na strukturách zásobník



Obrázek 4.1: Ukázka stromu webových stránek

a datové pole. V zásobníku se ukládají URL pro další provádění a úroveň zanoření (je možné pomocí parametru *-l N*, kde N je hloubka zanoření, definovat, do jaké „hloubky“, se budou odkazy prohledávat). Na referenčním obrázku 4.1 vidíme, že zanoření úrovně 0 je stránka *http://unicv.zcu.cz*, zanoření úrovně 1 jsou podstránky */prihlaseni, /registrace, /vyhledavani, /kontakt* a tak dále.

4.3.1 Funkce zásobníku



Obrázek 4.2: Ukázka funkce zásobníku a archívu

Do zásobníku jsou ukládány instance třídy *Parser::StackItem* (tato třída je pouze „přepravka“[1]), které mají 2 atributy: URL a hloubku zanoření. Před samotným spuštěním procházení je do zásobníku vložena kořenová stránka (v našem případě *http://unicv.zcu.cz*). Následně je spuštěn cyklus, který běží dokud zásobník není prázdný. Pokud stránku zpracujeme (včetně odkazů a formulářů), je uložena do pole historie. Každá nově přidávaná stránka do zásobníku je ověřována proti zásobníku i proti historii, aby nebyla jedna stránka procházena 2x.

4.3.2 Získávání a uchovávání odkazů

Při vytváření URL stromu webových stránek se ihned prochází načtené stránky a zjišťují se formuláře a odkazy na dané stránce. Pro uchování a následné

[1]Třída, která slouží pouze k uchování dat.

zpracování se využívají 2 pomocné třídy:

- *Parser::AContainer* - uchovává odkazy a parametry
- *Parser::FormContainer* - uchovává formuláře, metodu odesílání a jejich parametry

Při nalezení nového formuláře nebo odkazu jsou nejprve data porovnávána s poli ve třídě. Pokud již pole obsahuje formulář nebo odkaz, jsou tyto rozšiřovány, což je vidět u příkladu 4.3, který ukazuje jednoduchost porovnání (za zmínku stojí i aliasování metod, které je řešeno takto: *alias eql? ==*).

Kód 4.3: Porovnání dvou instancí třídy FormContainer

```
# Equals method for comparing
# @param [FormContainer] another_form_container Another form container
# @return [boolean] True or false
def ==(another_form_container)
   # Action URL
   return @action != another_form_container.action && @type !=
       another_form_container.type &&  @params != another_form_container.
       params
end

# Alias for ==
alias eql? ==
```

4.3.3 Normalizace URL

Na webové stránce máme relativně hodně možností jak zapisovat různé odkazy, od relativních cest *./*, přes absolutní *http://server.cz/index.php?action=nova*, až k pouhým „skokům“ na stránce *#novinky*. Veškeré tyto URL je potřeba normalizovat, ověřit server (pokud odkazy směřují mimo naší doménu, nejsou použity dále), získat data a odstranit nepotřebné části URL adresy. Při spuštění skriptu můžeme definovat „wildcardování“ domén, což znamená, že zadáme doménu prvního řádu: *http://zcu.cz* a pokud je povoleno, bude skritp indexovat i domény vyšších řádů, například *http://unicv.zcu.cz*. V následující tabulce je ukázka případů, kde kořenová doména je *http://unicv.zcu.cz/*:

URL v odkazu	Normalizovaná URL
./index.php?action=help	http://unicv.zcu.cz/index.php?action=help
#novinky	http://unicv.zcu.cz/#novinky
unicv.zcu.cz/index.php?action=user	http://unicv.zcu.cz/index.php?action=user
http://www.seznam.cz/	**Chybná URL - mimo zadaný server**

Tabulka 4.1: Příklady normalizování URL

Prvním krokem bylo nalezení již hotového řešení a výsledek byl překvapující. Ruby po provedení výchozí instalace obsahuje knihovnu *uri*, která umožňuje požadované věci a umí s URL i velice snadno pracovat:

Kód 4.4: Normalizování URL pomocí třídy URI

```
# Spojeni 2 URL
url = URI.join("./index.php?action=help#fragment", "http://unicv.zcu.cz/")
# http://unicv.zcu.cz/index.php?action=help#fragment

# Odstraneni fragmentu
url.fragment = nil
# http://unicv.zcu.cz/index.php?action=help

# Ziskani parametru
url.query
# {:action => "help"}
```

4.4 Získání URL vectoru stránky

Pro porovnání délky / obsahu stránek jsem vytvořil modul *URLVector* (modul není třída), který má tři funkce:

1. Získání vektoru z HTML stránky
2. Odečtení dvou vektorů
3. Získání váhy vektoru

URL vektor je běžná *Hash*[2] struktura s pevně danými klíči, které jsou názvy elementů viz ukázka html stránky 4.5 a vektoru 4.6 z ní získaného.

Kód 4.5: HTML stránka

```
<html>
   <head>
      <title>Music blog</title>
```

[2]Klíč - hodnota, v Ruby se klíči říká *symbol*, který vždy začíná „:“

```
    </head>
    <body>
        <h1>My music blog</h1>
        <div class="song">
            <strong>Song name</strong>
              ...
        </div>
          ....
    </body>
</html>
```

Kód 4.6: Získaný URL vektor

```
url_vector = {
     :html => 1, :head => 1, :title => 1,
     :body => 1, :h1 => 1, :div => 50, :strong => 50
}
```

4.5 Testování se zapnutými chybovými direktivami pro PHP

Testování se zapnutými chybovými direktivami je postaveno na principu, že se do parametrů postupně přidá uvozovka, která „zničí“ SQL dotaz, v tomto dotazu bude tudíž uvozovka navíc a tento dotaz se nepodaří provést a PHP preprocesor zahlásí chybu. Na stránce jsou tyto chyby následně vyhledávány:

- do verze PHP 5.2 - mysql_error
- od verze PHP 5.2 - php notice pro nesprávné použití *while* (v konstrukcích iterací výsledky) nebo pro přístup k asociovaným polím, které neexistují.

Pokud je nějaká tato chyba na stránce nalezena, existuje zde vysoká pravděpodobnost, že se podařil SQL injection a tudíž byl SQL dotaz modifikován.

4.6 Testování obsahu stránky při výměně parametrů

Testování bez zapnutých direktiv není jednoznačné a výsledky je třeba ověřit ještě ručně. Pro porovnání výsledků s nahrazením parametrů se používají 3 možné případy:

1. ’ - „rozbití“ SQL dotazu
2. ’ – - zakomentování zbytku SQL příkazu

3. *' OR '1' = '1* - logická pravda

4.6.1 Testování výsledků formulářů

Formuláře se obecně používají na běžných webových stránkách ke 3 účelům:

1. Přihlašování
2. Vyhledávání
3. Filtrování

Pokud se „podvodný“ vstup dostane do dotazu, bude mít u každého typu formuláře jiný výsledek, proto se musí otestovat stránka jinak. Máme tudíž 4 webové stránky a vektor prvků na stránce. Vezmeme-li si možné situace, uvidíme, co se s vektory bude dít oproti bezchybnému průběhu. (Pozn. Body odpovídají výměně parametrů v sekci *Testování a výměna parametrů*)

- **Přihlašování**
  1. Snížení počtu prvků na stránce, často přesměrování na nový formulář s hláškou, že chybný požadavek nelze zpracovat, protože SQL dotaz nebylo možno provést.
  2. Velká změna při nahrazení v položce „jména“. Díky zakomentování dotazu bude uživatel autorizován a bude přesměrován na stránku s jiným obsahem, zbytek SQL dotazu bylo zakomentováno popřípadě nastala logická pravda, tudíž vždy vrací výsledek.
  3. To samé jako v bodě 2)
- **Vyhledávání a filtrování**
  1. Snížením počtu prvků na 0 dotaz nevrátí žádný výsledek = snížení počtu elementů stránky.
  2. Zvýšení počtu elementů, může být dosti zásadní (vypisují se všechny prvky) nebo menší (přibudou položky ve stránkování).

Z výše zmíněného vyplývá, že je třeba určit všechny možnosti a určit případy značnými rozdíly dle modelu chování, protože nejsme schopni strojově rozpoznat, jaký formulář testujeme.

4.6.2 Testování výsledků odkazů

Odkazy se testují na stejném principu jako formuláře, tj. hledají se změny obsahu pomocí vektoru elementů na stránce.

4.7 Insert-into metoda

Další možná metoda zjištění SQL injection, tato metoda vyžaduje předchozí zásah administrátora nebo vývojáře, je totiž nucen předem vytvořit tabulku *sid_log* (jejíž definice je uložena ve složce *sql_dump*). Po spuštění testu je do každého dotazu přidávána struktura *'; INSERT INTO sid_log(param) VALUE(param_name).* Tudíž pokud dotaz projde, je do tabulky přidáno jméno parametru, který tento vstup umožnil. Odpovědné osobě následně stačí tuto tabulku projít po dokončení testu.

4.8 Drop-All metoda

Do každého dotazu je přidána direktiva *'; DROP ALL tables; '* – nebo *'; TRUNCATE ALL tables; '* –, která způsobí vymazání všech tabulek. Pokud tento dotaz projde, změna obsahu bude skutečně zásadní, popřípadě bude zobrazena stránka 500 (služba je nedostupná). Tato metoda je *opravdu destruktivní*, protože smaže opravdu vše.

Kapitola 5

Porovnání s existujícími nástroji

5.1 Jak probíhalo testování?

Testování probíhalo na vzorové stránce, která byla připojena k databázi s katalogem hudby (viz kapitola 3). Testovací stránka obsahovala 4 formuláře různých typů a odkazy. Polovina parametrů byla nezabezpečených, tudíž mohli být napadnuty.

5.2 OWASP ZAP

5.2.1 Co je OWASP ZAP?

The Open Web Application Security Project je celosvětová nezisková charitativní organizace[1], jejíž cílem je zvýšit zabezpečení softwarových řešení. Veškeré jejich snahy jsou cíleny na zviditelnění možností bezpečnostních opatření. Všechny materiály, knihy a software, který je vyvíjen je zdarma.

Obrázek 5.1: https://www.owasp.org/

5.2.2 OWASP ZAP

OWASP Zed Attack Proxy Project je penetrační test pro nalezení bezpečnostních chyb ve webových aplikacích, umožňuje manuální kontrolu i automatické testy. Je k dispozici zdarma na: *https://code.google.com/p/zaproxy/*.

5.2.3 Nalezené výsledky

OWASP na testovací stránce nezjistil vůbec žádné problémy typu SQL injection, i když byly zapnuty chybové direktivy. Detekoval ovšem dalších mnoho problémů s procházením adresářů či špatném zasílání hlaviček.

5.3 SQLMap

SQLMap - Automatic SQL Injection and database takeover tool, tato utilita byla zajímavější než výše předchozí OWASP. Je napsána v jazyce Python a je zdarma k dispozici na *http://sqlmap.org/*. Dokáže přes nezabezpečený parametr získat veškeré informace o databázovém serveru, vytvořit uživatele (je-li to možné) nebo získat struktury tabulek celé databáze.

5.3.1 Nalezené výsledky

Nepodařilo se mi realizovat, aby SQLMap stránky prohledával sám (jako to dělá OWASP), ale při nalezení nezabezpečeného vstupu se ukázal jako výborný pomocník (jak už název napovídá) k převzetí databáze, což se na nezabezpečeném vstupu povedlo bez problémů.

5.4 OWASP / SID + SQLMap

SQLMap a OWASP / SID dohromady tvoří zajímavý nástroj pro detekci a následné „zneužití“ SQL injection chyby. OWASP by byl využit pro zjištění všech možných vstupů, které by následně SQL Map otestoval, které je značné množství a SQL Map provádí testy opravdu dlouho. Při využití SIDu a SQL Mapu bude tento čas menší. Při testech SID + SQL Map na vlastním serveru jsem získal všechny tabulky a data z nich.

5.5 Acunetix - Web Vulnerability Scanner

Tento nástroj nebyl v testu použit, protože je placený a není k dispozici pro jiné operační systémy než je Microsoft Windows. Dle informací od výrobce (*http://www.acunetix.com/*) by měl být schopen automaticky detekovat jak SQL Injection tak XSS problémy.

Kapitola 6

Ukázky použití SID

6.0.1 Stažení a instalace

SID je možné se zdrojovými kódy stáhnout z GIT repozitáře, který je umístěn na serveru GitHub, kde byl v rámci bakalářské práce vyvíjen. Repozitář je read-only, takže je možné ho standardní cestou naklonovat. Dále je přiložen soubor *Gemfile* a *Rakefile*. První ze souborů umožňuje automatické stažení potřebných balíčků (tzv. *Gemů*) pro spuštění programu. Druhý (*Rakefile*) slouží pro automatické spuštění testů ze složky *test*. Celý postup je uveden v následující části 6.1.

Kód 6.1: Instalace

```
# Klonovani repozitare
git clone git@github.com:Strnadj/SID.git SID

# Presun do slozky penetracniho testu
cd SID/PenTest

# Stazeni potrebnych balicku
bundle

# Spusteni vsech testu - volitelna moznost
rake

# Presun do slozky se spustitelnou binarkou
cd bin/

# Spusteni testu
./pentest
```

Pokud vše proběhlo správně, zobrazí se nápověda v příkazové řádce s možnými parametry. Jediný povinný parametr je *-u*, kterým zadáváme URL, která se bude zpracovávat. Ostatní parametry jsou nepovinné.

6.0.2 Ukázka spuštění

Pro základní spuštění použijeme následující příkaz:

Kód 6.2: Spuštění testu

```
./pentest -u http://localhost/test/ -d true -e false
```

Používáme tři parametry:

1. *-u* - URL testované stránky
2. *-d* - tzv. *debug/verbose* mód, který vypisuje veškeré informace o probíhajícím testu
3. *-e* - nastavení zobrazování chyb na serveru, pokud uvedeme false tak jsou direktivy vypnuté a porovnává se obsah

Pokud test spustíme bez parametrů zobrazí se nápověda se seznamem parametrů, které můžeme použít a jejich popisem.

6.0.3 Identifikace podezřelé proměnné

Pokud test nalezne proměnnou, která je podezřelá na SQL injection, je přehledně vypsána i s testovacími URL, které byly použity. Administrátor následně může tyto URL zkontrolovat a zkontrolovat parametry (pro identifikaci ve zdrojových kódech můžeme využít například příkaz grep).

```
= Start testing: =
Test links:
        Test: http://localhost/test/index.php
                Param: catId - probably UNSECURED - different results!
                        http://localhost/test/index.php?catId=1%27
                        http://localhost/test/index.php?catId=1%27+--+
                        http://localhost/test/index.php?catId=1%27+OR+%271%27+%3D+%271
Test post forms:
        Test: http://localhost/test/index.php
                Param: gender - probably UNSECURED - different results!!
```

Obrázek 6.1: Úspěšné nalezení nezabezpečeného parametru

Kapitola 7

Závěr

Cílem práce bylo získat dostupné informace o problematice bezpečnosti webových stránek a aplikací a vytvořit nástroj pro penetrační testování (SID) jednoho hlavního bezpečnostního rizika SQL Injection.

SID umožňuje 4 metody testování SQL injection: 1) Testování se zapnutými chybovými direktivami 2) Testování obsahu při výměně parametrů 3) Insert-into metoda a 4) Drop-all metoda. Využíváme-li metodu číslo 2, tak i přes pokus o zobecnění chování změn parametrů, které jsou náchylné k SQL injection, není bohužel možné zcela jistě ve všech ohledech říci, že tyto parametry jsou nechráněné. Vždy získáváme pouze podezření na SQLi.

Webových aplikací a stránek je nepřeberné množství a každá se chováním trochu odlišuje. Při testování na vlastních testovacích stránkách a na vzorové stránce společnosti Acunetix (*http://testphp.vulnweb.com/*) se podařilo detekovat značné množství výskytů SQL injection. Bohužel ale stačí pouze jeden neochráněný vstup. Proto bych při skutečném testování použil kombinaci všech možných dostupných nástrojů (zvažoval bych i zakoupení komerčních nástrojů), manuální procházení zdrojových kódů (minimálně částí, týkajících se zpracování těchto požadavků).

Nástroj pro penetrační testování bych rád vyvíjel dále, jedním z dalších rozšíření by bylo procházení JavaScriptových souborů pro hledání Ajaxových požadavků a jejich testování a testování při aktivním přihlášení.

Na několika testovacích webových aplikací se povedlo najít vytvořeným nástrojem všechny problémy oproti zbylým testovaným nástrojům.

Ze subjektivního hlediska byla práce velice zajímavá a přínosná nejen z pohledu bezpečnosti, ale i z hlediska jazyka ruby a parsování stránek. Vyvinutý nástroj pro penetrační testování je určen pouze pro testovací účely, jako autor se zříkám veškeré zodpovědnosti při použití k jiným účelům.

Literatura

[1] *OWASP community* **OWASP Wiki** [on-line]
`https://www.owasp.org/`, 2013

[2] *Komunitní server* **Soom.cz - Pokročilé techniky XSS** [on-line]
`http://www.soom.cz/index.php?name=articles/show&aid=485&title=Pokrocile-techniky-XSS`, 2008

[3] *Robert Auger* **Cgisecurity.com** [on-line]
`http://www.cgisecurity.com/csrf-faq.html`, 2010

[4] *OWASP comunity* **OWASP Wiki - Path traversal**
`https://www.owasp.org/index.php/Path_Traversal`, 2013 [on-line]

[5] *Ruby comunnity* **Ruby doc** [on-line]
`http://www.ruby-lang.org/en/documentation/`, 2013

[6] *Michael Santo* **Examiner.com** [on-line]
`http://www.examiner.com/article/lulzsec-posts-email-addresses-passwords-for-26-000-porn-site-users`

[7] *Zed A. Shaw and Rob Sobers* **Learn Ruby The Hard Way rev.2**
`http://programming-motherfucker.com/`, 2012 E-book

[8] *Joel Scambray, Stuart McClure, George Kurtz* **Hacking bez tajemství**
Computer Press, 2010

[9] *The PHP Group* **PHP - Documentation** [on-line]
`http://php.net/manual/en/`, 2013

[10] *Offensive Security* **Exploit-Db** [on-line]
`http://www.exploit-db.com/webapps/`, 2013

[11] *Neil Daswani, Christoph Kern and Anita Kesavan* **Foundations of Security** [on-line]
Apress, 2007

[12] *Charles Arthur, Dan Sabbagh and Sandra Laville* **The Guardian**
`http://www.guardian.co.uk/technology/2012/mar/06/lulzsec-sabu-working-for-us-fbi`, 2012 [on-line]

Seznam obrázků

Seznam tabulek

Seznam algoritmů