ISSN 2464-4617 (print)
ISSN 2464-4625 (DVD)
Computer Science Research Notes
CSRN 2901
WSCG Proceedings Part I

SyCaT-Vis: Visualization-Based Support of Analyzing System Behavior based on System Call Traces

Alrik Hausdorf[1]
hausdorf@informatik.uni-leipzig.de

Nicole Hinzmann[1]
hinzmann@informatik.uni-leipzig.de

Dirk Zeckzer[1]
zeckzer@informatik.uni-leipzig.de

[1]Image and Signal Processing Group,
Leipzig University

ABSTRACT

Detecting anomalies in the behavior of a computer system is crucial for determining its security. One way of detecting these anomalies is based on the assessment of the amount and sequence of system calls issued by processes. While the number of processes on a computer can become very large, the number of system calls issued during the lifespan of such a process and its subprocesses can be humongous. In order to decide whether these anomalies are due to the intended system usage or if they are caused by malicious actions, this humongous amount of data needs being analyzed. Thus, a careful analysis of the system calls' types, their amount, and their temporal sequence requires sophisticated support. Visualization is frequently used for this type of tasks. Starting with a carefully aggregation of the data presented in an overview representation, the quest for information is supported by carefully crafted interactions. These allow filtering the tremendous amount of data, thus removing the standard behavior data and leaving the potentially suspicious one. The latter can then be investigated on increasingly finer levels. Supporting this goal-oriented analysis, we propose novel interactive visualizations implemented in the tool SyCaT-Vis. SyCaT-Vis fosters obtaining important insights into the behavior of computer systems, the processes executed, and the system call sequences issued.

Keywords

Security visualization, system call traces, security analysis, behavior analysis

1 INTRODUCTION

Detecting anomalies in the behavior of a computer system is crucial for determining its security. The analysts need to decide if these anomalies are due to intended system usage or if they are caused by malicious actions. To do so, the behavioral analysis can be based on observing the system calls issued by processes like memory reads and writes, network usage, and CPU usage, among others. Besides the types of the system calls, their amount and their sequence are important clues for separating intended from malicious behavior.

Overall, this behavioral system analysis is quite involved due to the huge amount of individual system calls and the already large number of processes that run in parallel. To date, only some approaches for automated analysis of system call traces were proposed (Section 2.1). Regarding visual support of these analyses, only three approaches were found in the literature [16, 14, 2] (Section 2.2). Our contribution is the tool SyCaT-Vis (SystemCallTrace-Visualization) providing three views containing interactive visualizations adapted to this situation:

1. A Context View showing all traces that were captured.
2. A Process view proving more detailed, temporal information about programs and program groups.
3. A detailed thread view showing groups of system calls as well as individual system calls themselves.

This realizes a semantic zooming environment that together with filtering as well as additional interactions fosters the analysts insights into the process behavior recorded and thus eases analyzing the security of the computer system under investigation (Section 4).

2 RELATED WORK

2.1 Automated Analysis of System Call Traces

Analyzing the behavior of a system is often performed based on data about which processes are running and

Data Set	t	CA	E	SC	SCT	SCC	C	U	P	T
1	1s	i	21,362	10,756	23	9	4	6	22	75
2	5s	i	62,353	31,339	72	12	1	5	84	143
3	8s	i	105,384	52,900	81	12	1	7	101	165
4	∼60s	i	784,167	392,896	88	12	1	7	304	516
5	∼27s	au	3,674,663	1,838,122	106	13	7	14	145	715

Table 1: Basic values of the benchmark data (t: time span; CA: computer activity (i: idle, au: actively used); and the contained number of elements (see also Table 2; E: events (without switch events); SC: system calls; SCT: system call types; SCC: system call categories; C: containers; U: users; P: processes; T: threads)

Information Type	Result-Set	sysdig Parameters
event information	E_{inf}	evt.num, evt.rawtime, evt.latency, evt.dir
system call type	S_{type}	syscall.type
event category	E_{cat}	evt.category
container	C	container.id, container.name
user	U	user.uid, user.name
process	P	proc.pid, proc.exe, proc.args, proc.pname, proc.name
thread	T	thread.tid

Table 2: Attributes extracted from the complete sysdig trace.

which system calls are issued by them. To obtain low false-positive rates in detecting malicious system behavior, several different models supporting the automated analysis of system call traces have been developed.

Forrest et al. [6, 4, 5] developed an n-gram pattern approach. Sequences of system calls are analyzed and those stemming from normal process execution are used to form pattern that are stored in a database. The system call sequences to be analyzed are then compared to the pattern stored in the database and mismatches are reported. Sekar et al. [10] and Yu et al. [17] use finite state automata instead of the n-grams for this analysis. Ghosh et al. [7] use neural networks to determine whether or not the system behavior is malicious, while Liao et al. [8] use k-nearest neighbor classification for pattern detection overcoming the need of learning sequence pattern for individual programs.

Warrender et al. [15] compared several methods to detect intrusions: enumeration of observed sequences, a rule induction technique, and Hidden Markov Models. They conclude that "weaker methods than Hidden Markov Models are likely sufficient" for these tasks.

Coull et al. [1] use an approach that is based on techniques used in bioinformatics to uncover masquerade attacks. The trace to be analyzed is compared to previous traces using semi-global alignment. A similarity score is computed and based on this score the similarity or dissimilarity between the traces that are compared can be assessed.

Mazeroff et al. [9] employ an approach based on probabilistic models. They construct probabilistic suffix trees and translate them into probabilistic suffix automata. The resulting models are then used for monitoring data in real-time.

Shu et al. [13] analyze the correlation among events using long-span behavior anomaly detection based on mildly context-sensitive grammar verification.

All these approaches rely on machine learning approaches, generating models and assessing the systems' behavior according to these models. No visual support for analyzing the results is provided.

2.2 Visual Analysis of System Call Traces

Visualizations supporting system behavior analysis based on system call traces are less common. Wu et al. [16] propose "lviz", a tool visualizing Microsoft Windows system call traces. Their visualization shows two call traces and the contained events in a dotplot matrix. Tandon et al. [14] visualize distances of motifs found in system calls using scatterplots. Lately, the tool Csysdig [2] showing the latency of system calls per time in a spectrogram was proposed.

Our tool provides three visualizations allowing on the one hand to obtain a general overview and on the other hand analyzing and comparing several process execution traces at the same time. Moreover, the user can smoothly change between these views in a top-down or bottom-up manner according to her workflow.

3 DATA SETS

For testing the performance and showing the features of our visualization, we use self-generated data sets (Table 1). The behavioral data was collected using the tool "sysdig" [3]. This tool does not provide system calls directly. Therefore, these have to be extracted from the

 ISBN 978-80-86943-37-4

behavioral data comprising the events associated with the system calls. Sysdig stores the data obtained in compressed, binary files, the proposed file extension being “scap”.

As the amount of attributes collected by sysdig is very large, it is useful to extract those attributes that should be analyzed before subsequent operations. This is done calling sysdig using a set of parameters indicating the attributes to be extracted. The information we are interested in together with the corresponding sysdig parameters is given in Table 2.

"switch"-events are not related to system calls and just indicate a context switch, i.e., when a thread is put to sleep by the process scheduler, while another thread will be executed. Therefore, they are filtered from the data. Data selection and filtering are performed while loading the original data into our tool.

Data Set 5 contains the largest amount of collected system calls and therefore will be used as example for presenting our approach.

4 SYCAT-VIS

4.1 System Overview

To support analyzing a system’s behavior based on its system call traces obtained using sysdig [3], we propose SyCaT-Vis (SystemCallTrace-Visualization). All data to be analyzed is stored in a PostgreSQL database. SyCaT-Vis imports the data collected into this database by running the sysdig command as described in the previous section for selecting and filtering the original system traces (scap-file) followed by parsing the results and writing them into the database. The importer itself is written in the Java programming language.

A ReSTful API written in NodeJS connects the SyCaT-Vis user interface to this database. It also serves as a cache for database requests needing a long computing time. The user interface itself is written in JavaScript using the AngularJS library (version 1.7.7) for data and interaction handling, and the D3.js library for creating the visualizations.

Currently, three views are provided by the user interface for analyzing the system behavior following Shneiderman’s mantra “Overview first, zoom and filter, then details on demand” [12]. The Context View (Section 4.2) provides an *overview*: information about the main attributes and their values in the information areas, a user-selected hierarchy of attributes in the main area, and a configuration area for selecting and deselecting the attributes and their hierarchy shown in the main area realizing a *filter*. A contextual popup-menu provides additional information (*details on demand*) and allows using the currently selected element as a *filter* in the Process View.

The Process View (Section 4.3) is a more detailed, *intermediate view* (*zoom*) showing the time evolution of all processes or of those processes that match the filter conditions selected. Various interactions allow modifying the display of the information as well as adapting the *filters*. Altogether, the filter conditions can be selected in the context view and they can be selected and changed in the process view.

Selecting individual processes in the Process View can in turn be used as a filter for the most *detailed view*, the Thread View (Section 4.4). This view shows the system calls for the selected threads of a process from a coarse grained aggregated view summarizing system calls to a fine grained view showing individual system calls, thus realizing several zooming levels.

We describe these views in the subsequent sections using Data Set number 5 (Table 1) as a running example.

4.2 Context View

The *main area* of the context view (Figure 1) contains a circle-based hierarchical view showing an overview of the data (Figure 1 (c), Section 4.2.3). It is complemented by a *configuration area* allowing selecting the elements displayed in the hierarchy and their order (Figure 1 (b), Section 4.2.2). Moreover, the configuration area provides selecting scaling the attribute values being visualized linearly or logarithmically. The *information areas* to the left and to the right of the visualization and interaction areas provide information about the entries for the main attributes (container: upper left, users: lower left, processes: upper right, and system call types: lower right) (Figure 1 (a), Section 4.2.1). Finally, a *contextual popup menu* provides further interaction facilities (Figures 1 (d) and 2, Section 4.2.4). Next, we describe each of these areas in the order of the work flow of an end user.

4.2.1 Information Areas

The *information areas* (Figure 1 (a)) are to the left and to the right of the visualization and interaction areas providing information about the different values of the main attributes. These correspond to the previously (Section 3) defined sets C: Container (upper left), U: Users (lower left), P: Processes (upper right), and S_{type}: System Call Types (lower right). For each of these attributes, its name and the amount of different values are given. Additionally, the eye-icon indicates that this dimension is currently part of the hierarchy displayed in the main area of this view. The attribute values are sorted by the amount of system calls they are connected to in descending order. The top eight attribute values according to this order are shown together with the amount of system calls they are related to. Finally, a hint about how many more attribute values are not shown is displayed. On mouse-over one of the attribute

 ISBN 978-80-86943-37-4

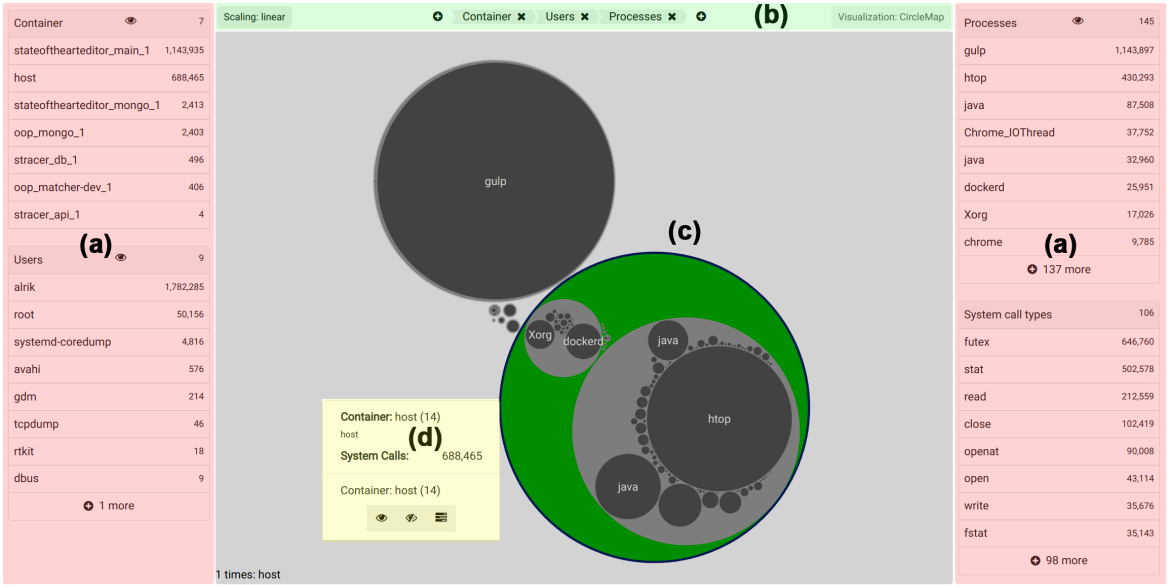


Figure 1: The Context View containing the following areas: (a) Information Areas showing the attributes that can be included into the hierarchy (b) shown in the main area (c); (b) Configuration Area allowing to construct the hierarchy to be analyzed as well as to switch between linear and logarithmic scaling; (c) Main Area showing a circle-based hierarchical visualization representing the currently active attribute hierarchy; (d) Context Menu showing information about the selected element (green) and providing interaction facilities.

values, all occurrences of this attribute value in the main area are highlighted, if this attribute is shown there.

In our example, the "Container", the "Users", and the "Processes" are part of the hierarchy shown in the middle (having an eye icon), while the "System Call Types" are not (no eye icon). All container attribute values are shown (7) in its list, while only 8 out of 10 users (2 more not displayed), 8 out of 145 processes (137 more not displayed), and 8 out of 106 system calls types (98 more not displayed) are listed in the respective areas. User "alrik" is associated with the largest number of system calls (1,782,285), followed by "root" (50,156).

4.2.2 *Configuration Area*

The configuration area is located between the two information areas at the top (Figure 1 (b)). In the middle of the configuration area, the attributes currently shown in the main area are displayed in the order in which they form the hierarchy from top (left) to bottom (right). Each of the currently shown attributes can be removed. Moreover, new attributes can be added to the top or to the bottom of the hierarchy.

On the left side of the configuration area, the user can switch between linear and logarithmic scaling of the attribute values visualized in the main area. Logarithmic scaling is especially useful if small elements should be made more visually salient.

In the example given in Figure 1, the hierarchy is "Container" (top) – "Users" – "Processes" (bottom). Moreover, linear scaling is chosen.

4.2.3 *Main Area*

In the main area of the context view (Figure 1 (c)), the currently selected hierarchy is displayed using a circle-based hierarchical view. The circles are color-coded according to the hierarchy they belong to, the lightest color representing the top attribute of the hierarchy and the darkest color representing the bottom attribute of the hierarchy. Moreover, the size of each circle reflects the number of the associated system calls.

In the example, the basic color is gray. Thus, the seven values of the top-most attribute of the hierarchy ("Con-

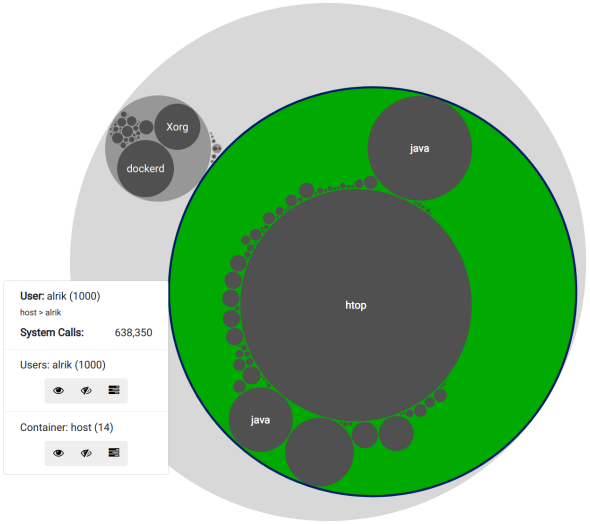


Figure 2: Context View restricted to the container "host" with user "alrik" being selected.

ISBN 978-80-86943-37-4

tainer") are mapped to seven light-gray circles that are arranged next to each other in the main area. The values of the next attribute in the hierarchy ("Users") are grouped by container value and each container-users pair is mapped to a middle-gray circle. These circles are arranged inside the circles representing their respective containers. Finally, the values of the bottom-most attribute of the hierarchy ("Processes") are grouped by container-users value pairs yielding container-users-processes triplets. Each such triplet is mapped to a dark-gray circle. These circles are arranged inside their respective container-users circles.

If an attribute value corresponding to a circle is selected, the circle is colored green. In the example, the green circle represents the container "host". Within this container, there are circles for all users active in this container (eight). Some of these circles are very small and can barely be seen. Logarithmic scaling could be used to enlarge them when needed. Within each of the circles representing the users, the dark circles represent the processes like the two "java" processes and the single "htop" process inside the circle representing the lower right user of the container "host", or the process "gulp" on the upper left.

Figure 2 shows the main area restricted to the container "host" with user "alrik" being selected (circle colored green). Only the main area is shown.

We also tested a squarified treemap [11] representation of the same data (see also `http://www.cs.umd.edu/hcil/treemap-history/`). In general, a treemap is emphasizing leaf information over hierarchy information. Also, treemaps are prone to create very small, sub-pixel wide or high stripes. Comparing the circle layout to a squarified treemap layout showing the same data (Figure 3), it can be seen that the hierarchy is more salient and that the processes with less activity are easier to discern in the circle representation (Figure 3(a)) compared to the squarified treemap representation (Figure 3(b)). Further, it can be seen, that the unused space around the circles helps to spot elements that are very small and that can not be identified in the squarified treemap representation. As both the hierarchical information and the inner nodes (and information about them) are both important for our application, the circle-based hierarchical view was chosen.

4.2.4 Context Menu

Selecting a circle in the main area results in displaying a contextual popup menu (Figure 1 (d)), Figure 2) showing information about the element associated with the circle representing a specific attribute selected (Table 3 shows the values for the examples provided in Figure 1 (d) and Figure 2):

- The *name* of the *attribute*
- The *value* of the *attribute*

Information shown	Figure 1 (d)	Figure 2
Attribute Name	Container	User
Attribute Value	host	alrik
Element ID	14	1000
Hierarchy	host	host > alrik
System Calls Amount	1,375,457	1,275,438

Table 3: Attribute values provided by the popup menu shown in Figure 1 (d) and Figure 2, respectively.

- The *id* of this *element*
- The elements of the *hierarchy* starting at the top and ending at the selected element including the selected element
- The *amount of system calls* that are associated with the selected element

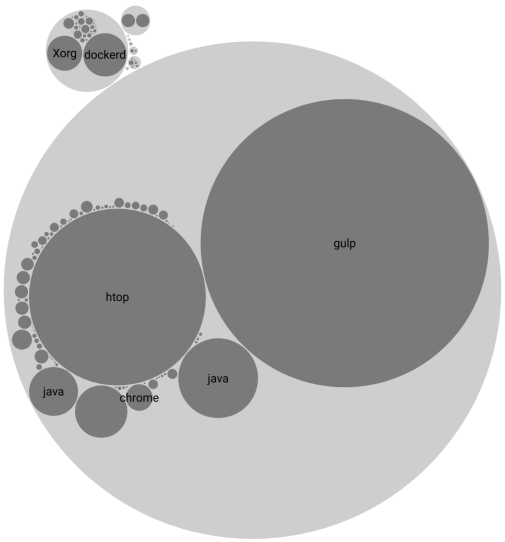
Moreover, the context menu provides several interaction facilities. For each hierarchy level above and including the current element, a filter can be selected at the bottom of the menu. The filter options are to select the element (none selected), to hide the element (strike-through eye, middle), and show only this one (eye, left). Each set of icons is annotated by the hierarchy information in the format "attribute: attribute value (id)". In Figure 1 (d), there is only one entry: "Container: host (14)", while in Figure 2, there are two entries: "Users: alrik (1000)" and "Container: host (14)" according to the two hierarchy levels involved in the selection. The third icon changes to the Process View (Section 4.3) showing the activities of the processes filtered by the selected element.

4.2.5 Interpretation

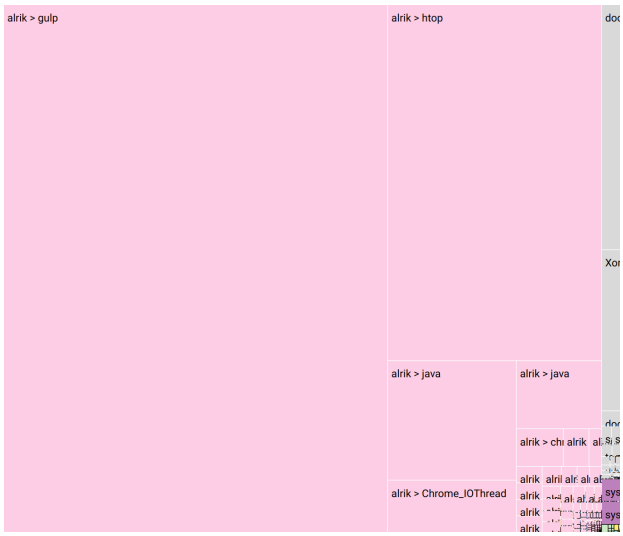
The Context View provides a general overview over the data set. Using this overview, first assumptions about the usage of the monitored computer are created. To do so, a solid knowledge is needed about which users are executing processes on this machine, which processes are expected to be running, and what the non-malicious behavior on this machine should look like. In the example shown in Figure 1, the existence of containers is expected on the machine under exam and thus should not cause an alert. However, a similar image based on the data drawn from an office only computer could be an indication of a misuse of the machine. Together with the activity of processes used by different users, this could indicate malicious behavior, e.g., by a malware infected computer.

On the other hand, if the inspected system is a server, the view of the container and the active processes inside is helpful. Based on the idea to have one container for one purpose, there should not be several processes that are equally active. If a container looks like the selected host of Figure 2, the assumption that the container was attacked could be made and that, e.g., a reverse shell

ISBN 978-80-86943-37-4



(a) Each user is represented by a light grey circle. The running processes of the user are represented by circles having a darker grey color located inside the user's circle. The size of the circles represents the amount of system calls aggregated for the respective circle. Compared to the treemap layout (b), the low activity processes are easier to discern and the hierarchy is salient.



(b) Squarified treemap representation of the "User" > 'Processes" hierarchy. Each user is mapped to an individual color. The size of the leafs representing the processes is mapped to the amount of their system calls. Compared to the circle layout (a), the low activity processes more difficult to discern and the hierarchy is less salient.

Figure 3: Comparison between the circle layout (a) and the squarified treemap layout (b) showing the Main Area of the Context View with no filters representing the hierarchy is "User" > "Processes".

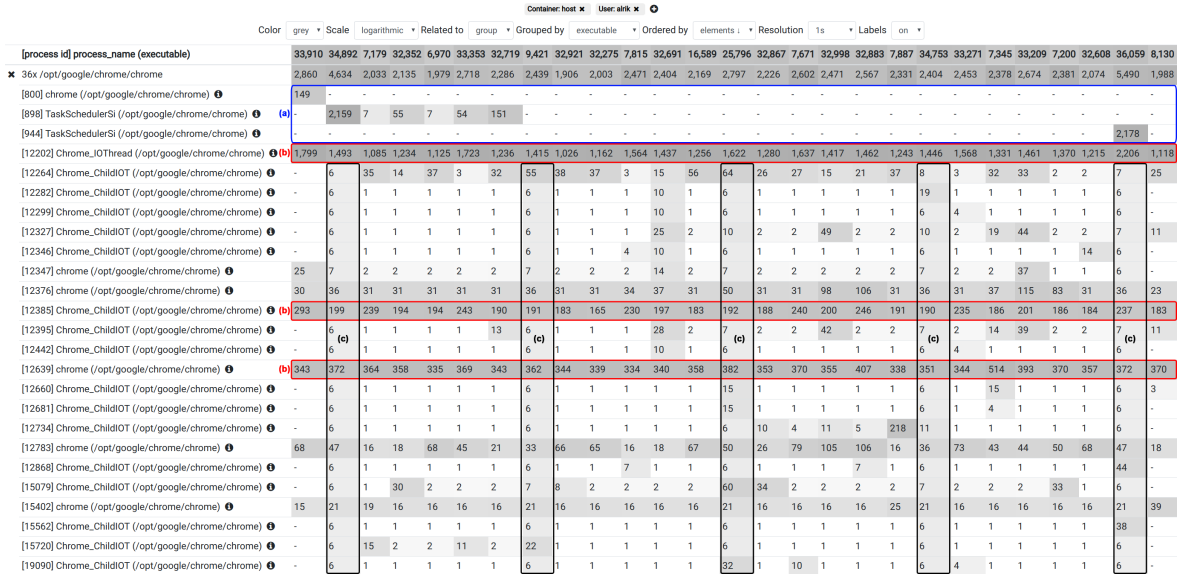


Figure 4: The Process View for user "alrik" in the container "host". The number of system calls per process overlapping a specific time step is encoded using a logarithmic grey scale. The relation is "group" and the processes are grouped by "executable". The groups are ordered by the count of "elements". The time resolution is one second.

was used to deploy malicious software inside a container.

4.3 Process View

The Process View (Figure 4) shows an overview of processes and their corresponding activities. Either all processes are shown or only the ones that were previously selected in the Context View. The filter currently applied to this view is shown at the top of the view. Filters can be removed and added there, too.

4.3.1 Design

A tabular design was chosen for displaying the processes and their evolution over time. Each process is

ISBN 978-80-86943-37-4

represented by a row and each time step is represented by a column. The number of system calls overlapping a time step is represented by the respective cell's lightness: the darker the cell, the more system calls are associated with the (sub-)process and overlap this time step.

By default, the processes are grouped by the "parent_name", a 16 bit identifier for the process. Every process in a group is spawned from the same named parent. It is possible that processes are spawned from different parent processes with the same parent_name. The parent_name should be a significant identifier for the activity or for the goal of the process. Therefore, the grouping of processes with similar named parents is interesting to us. Another option is to group the processes by the associated executable.

The processes can be ordered by the process id of the first system call in each group, or by the number of processes per group. The ordering can be either ascending or descending.

The labels shown in Figure 4 (process id, executable, and number of system calls) can be hidden such that only the visual elements are shown. Then, the information can be shown on mouse-over the respective cell.

Besides changing the filters, additional interaction facilities are provided by this view (Figure 4, below the filters; selected values are emphasized):

- Select the color hue ("Color": *grey*, blue, red)
- Select the scale ("Scale": linear, *logarithmic*)
- Select the relation ("Related to": *group*, global)
- Group by ("Grouped by": *executable*, parent_name)
- Order by ("Ordered by": *process id*, number of elements in group; *ascending*, descending)
- Select the time resolution ("Resolution": between 50ms and *1s*)
- Select if labels are shown ("Labels": *on*, off)

Here, the processes in the container "host" for user "alrik" are shown that are related to the executable "chrome". Overall, 25 processes and 27 time steps are shown in Figure 4.

Several pattern emerge from this visualization:

(a) The first three processes are different from the remaining ones. (see blue bordered area (a) in Figure 4)
(b) Several processes are constantly active, i.e., every time step. (see red bordered area (b) in Figure 4)
(c) Several processes show a regular activation (sub-)pattern. (see black bordered area (c) in Figure 4) They are at least active every six seconds.

Based on these pattern, hypotheses about the executable's behavior can be created.

4.3.2 Interpretation

Being an intermediate view between the Context View and the Thread View, the Process View provides the temporal dimension of active processes that are related to each other. One malicious behavior that is identifiable using this view is reverse-shells on servers. The possible indication for this type of malicious behavior is a long idle time followed by an extremely high amount of activity of another process directly afterwards, e.g., a php-child.

A similar example of this behavior—a process starting a new process that performs some sub-tasks—can be found in Figure 4 in the lines marked with an (a). In this case, the chrome process with process id 800 (line before) has no further activities after starting. However, directly afterwards, a high amount of system calls by its child process having the pid 898 is observed. Knowing the spawning schemata of specific processes, like using a fixed process-pool for computation, it can be assessed whether or not this behavior of creating new sub-processes and delegating tasks to them is malicious.

4.4 Thread View

4.4.1 Design

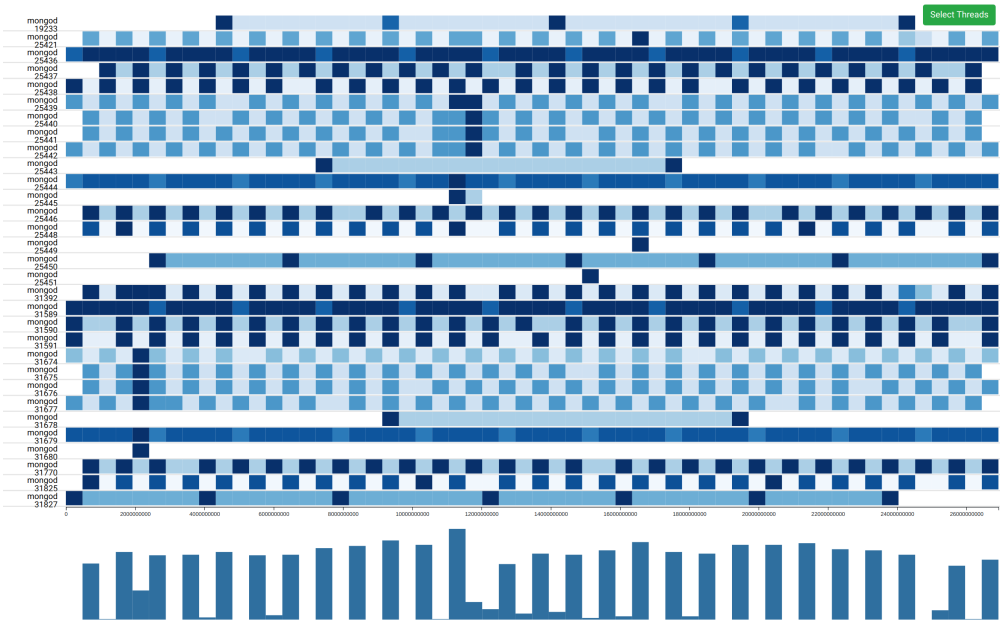
Figure 5 shows the system call traces at thread level. Each line—except the last two—represents a thread that can be selected from a list. In the last line, a barchart shows the distribution of systems calls over time. The number of all system calls from all threads overlapping the respective time interval is mapped to the height of the associated blue bar of the bar chart. On mouse over, the exact number of system calls overlapping each time interval is shown. Above this bar chart, the timeline is shown.

Figure 5(a) shows the coarsest granularity where the number of time intervals depends on the available screen space. Similar to the "Process View", a cell represents all system calls for a specific thread (row) overlapping a specific time interval (column). The saturation of the cell represents this number of system calls in relation to

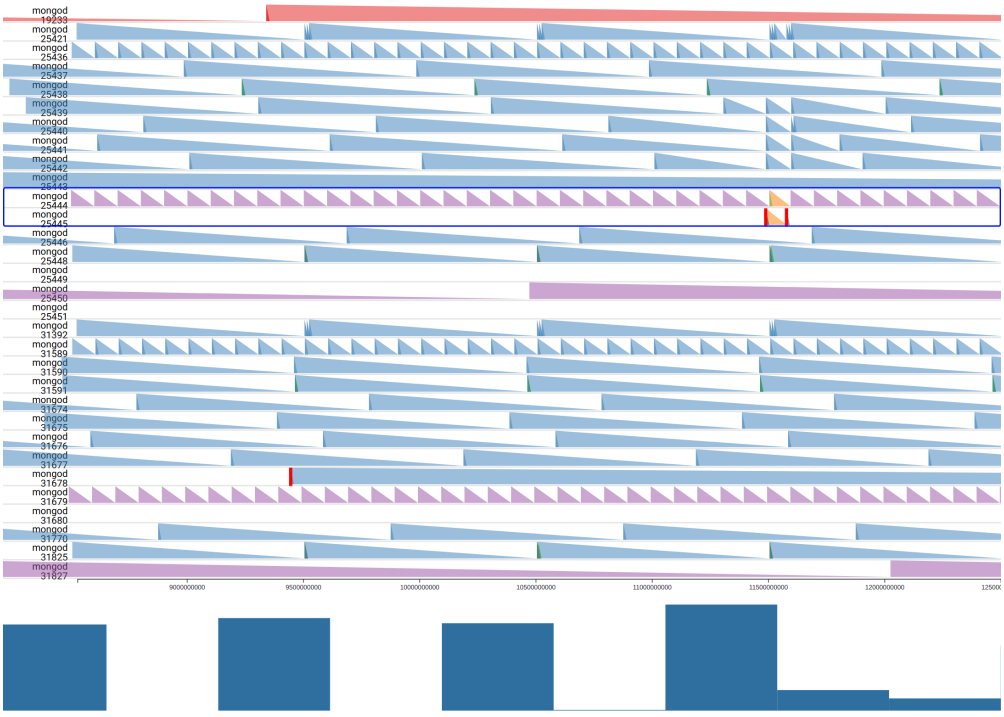
- the total number of system calls of the thread (used in Figure 5)
- the maximal number of system calls overlapping a time step over all threads
- the sum of system calls overlapping a time step over all threads
- the sum of system calls over all threads

Interaction allows to obtain more details. Zooming-in using the mouse allows to decrease the length of the time intervals. Thus, fewer, shorter time intervals are shown. These shorter time intervals are represented by cells as long as they contain more system calls than can be displayed. Otherwise, triangles and bars representing individual system calls are used. Each triangle starts at the beginning of the system call and ends at its termination. The triangle's color represents the system call category:

ISBN 978-80-86943-37-4



(a) Coarsest view showing the complete time interval selected. The saturation of each cell represents the number of system calls that are active in that interval: the higher the saturation the more system calls are contained in this thread and time interval.



(b) Finest view showing individual system calls, only. Triangles represent the system calls and their durations (called 'latency" by sysdig) from the start event to the final event of each system call. The triangle's color represents its system call category: *red*: net(work), *blue*: inter-process communication (IPC), *violet*: sleeping, *orange*: unknown (system call type: fdatasync). Single vertical red lines represent system calls with zero duration (e.g. in the marked lines, before and after the yellow system call). The irregularities in the idle threads (see marked lines) can easily be identified.

Figure 5: Visualization of the threads showing only aggregated system calls (a) and only individual system calls (b). In the last line, a barchart shows the distribution of systems calls over time. The number of all system calls from all threads overlapping the respective time interval is mapped to the height of the associated blue bar of the bar chart. On mouse over, the exact number of system calls overlapping each time interval is shown. Above this bar chart, the timeline is shown.

ISBN 978-80-86943-37-4

- *red*: net
- *blue*: inter-process communication (IPC)
- *violet*: sleeping
- *orange*: unknown (system call type: fdatasync).

A red bar represents a system call of zero duration. The finest level displaying individual system calls, only, is shown in Figure 5(b) (most detailed view).

This flexibility allows to generate as many intermediate levels between the coarsest (least detailed) and the finest (most detailed) level as necessary to provide an acceptable number of elements (bins or individual system calls).

4.4.2 Interpretation

The temporal pattern that can be observed in this visualization fosters recognizing normal as well as possibly malicious behavior. Again, a thorough understanding about the computer, its users, and the processes to be expected is needed for judging whether or not a certain visual pattern is suspected to be malicious or not. Therefore, the system calls that were issued are analyzed regarding their amount and their sequence. The detection of malicious behavior is one the one hand fosters by the side-by-side display of a number of processes performing the same task (see Figure 5 (b) line 4 to 9) such that processes with a divergent amount or sequence of system calls can be spotted. On the other hand, a sudden change in the behavior of a single process might point to a malicious irregularity. In Figure 5, the irregularities in the idle threads (lines in the blue bordered box, Figure 5) are an example of this type of anomaly.

5 CONCLUSION

We propose SyCaT-Vis for the visualization-based analysis of system call traces. Currently, three views are provided fostering understanding of the system's behavior: an *overview* showing the processes and their context (context view), an *intermediate view* showing details of selected processes (process view), and a *detailed view* showing details of selected threads of one or more processes (thread view, several levels of detail). Being able to configure and to interact with these flexible visualizations enables the security researcher to adapt them to her needs and to focus on the crucial parts for understanding whether the system's behavior is normal or not.

6 ACKNOWLEDGMENTS

This work was partially funded by the German Federal Ministry of Education and Research (BMBF) within the project Explicit Privacy-Preserving Host Intrusion Detection System EXPLOIDS (BMBF 16KIS0522K).

7 REFERENCES

[1] S. Coull, J. Branch, B. Szymanski, and E. Breimer. Intrusion detection: a bioinformatics approach. In *Proc. 19th Annual Comp. Security Appl. Conference*, pp. 24–33, Dec 2003. doi: 10.1109/CSAC.2003.1254307

[2] Draios Inc. *Csysdig Overview*, March 2017.

[3] Draios Inc. *Sysdig Overview*, March 2017.

[4] S. Forrest, S. A. Hofmeyr, and A. Somayaji. Computer Immunology. *Commun. ACM*, 40(10):88–96, Oct. 1997. doi: 10.1145/262793.262811

[5] S. Forrest, S. A. Hofmeyr, and A. Somayaji. The Evolution of System-Call Monitoring. In *Proceedings of the Annual Comp. Security Appl. Conference*, ACSAC '08, pp. 418–430. IEEE Computer Society, Washington, DC, USA, 2008. doi: 10.1109/ACSAC.2008.54

[6] S. Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff. A sense of self for Unix processes. In *Proceedings 1996 IEEE Symposium on Security and Privacy*, pp. 120–128, May 1996. doi: 10.1109/SECPRI.1996.502675

[7] A. K. Ghosh and A. Schwartzbard. A Study in Using Neural Networks for Anomaly and Misuse Detection. In *Proc. of the 8th Conference on USENIX Security Symposium - Volume 8*, SSYM'99, pp. 12–12. USENIX Association, Berkeley, CA, USA, 1999.

[8] Y. Liao and V. R. Vemuri. Using Text Categorization Techniques for Intrusion Detection. In *Proceedings of the 11th USENIX Security Symposium*, pp. 51–59. USENIX Association, Berkeley, CA, USA, 2002.

[9] G. Mazeroff, V. De, C. Jens, G. Michael, and G. Thomason. Probabilistic Trees and Automata for Application Behavior Modeling. In *41st ACM Southeast Regional Conference Proceedings*, pp. 435–440, 2003.

[10] R. Sekar, M. Bendre, D. Dhurjati, and P. Bollineni. A fast automaton-based method for detecting anomalous program behaviors. In *Proceedings 2001 IEEE Symposium on Security and Privacy. S P 2001*, pp. 144–155, 2001. doi: 10.1109/SECPRI.2001.924295

[11] B. Shneiderman. Tree Visualization with Treemaps: 2-d Space-filling Approach. *ACM Trans. Graph.*, 11(1):92–99, Jan. 1992. doi: 10.1145/102377.115768

[12] B. Shneiderman. The eyes have it: a task by data type taxonomy for information visualizations. In *Proc. IEEE Symp. on Visual Languages*, pp. 336–343, 1996. doi: 10.1109/VL.1996.545307

[13] X. Shu, D. D. Yao, N. Ramakrishnan, and

 ISBN 978-80-86943-37-4

T. Jaeger. Long-Span Program Behavior Modeling and Attack Detection. *ACM Trans. Priv. Secur.*, 20(4):12:1–12:28, Sept. 2017. doi: 10.1145/3105761

[14] G. Tandon, P. Chan, and D. Mitra. MORPHEUS: Motif Oriented Representations to Purge Hostile Events from Unlabeled Sequences. In *Proc. of the ACM Workshop on Visualization and Data Mining for Computer Security*, VizSEC/DMSEC '04, pp. 16–25. ACM, New York, NY, USA, 2004. doi: 10.1145/1029208.1029212

[15] C. Warrender, S. Forrest, and B. Pearlmutter. Detecting intrusions using system calls: alternative data models. In *Proc. of the IEEE Symp. on Security and Privacy (Cat. No.99CB36344)*, pp. 133–145, 1999. doi: 10.1109/SECPRI.1999.766910

[16] Y. Wu, R. H. C. Yap, and F. Halim. Visualizing Windows System Traces. In *Proceedings of the 5th International Symposium on Software Visualization*, SOFTVIS '10, pp. 123–132. ACM, New York, NY, USA, 2010. doi: 10.1145/1879211.1879231

[17] F. Yu, C. Xu, Y. Shen, J.-Y. An, and L.-F. Zhang. Intrusion detection based on system call finite-state automation machine. In *IEEE International Conference on Industrial Technology*, pp. 63–68, Dec 2005. doi: 10.1109/ICIT.2005.1600611