Benchmarking Virtualized Page Permission for Malware Detection: a Web Server Case Study

Pasquale Caporaso[1,2,*], Giuseppe Bianchi[1,2,*] and Francesco Quaglia[2,*]

[1]CNIT Natl. Network Assessment Assurance and Monitoring Lab, Rome, IT
[2]University of Rome "Tor Vergata", Rome, IT

Abstract

Operating systems are continuously offering new capabilities to detect malware. In this context, a new approach that has been recently proposed for the Linux world is based on virtualizing Write (W) and Execute (X) permission of pages in the address space of an application. This method has enabled the operating system kernel to determine any point in time where a page with updated content is accessed again by the CPU for instruction fetches. This offers new opportunities to track encrypted (packed) malware, which decrypts itself into writable/executable pages in the address space of an application. However, till today, the testing of this solution has been limited to desktop environments. In this article we perform benchmarking of WX permission virtualization in the context of a Web server system, providing the community with indications on the feasibility of this solution in service environments. The representatives of our study lies on the usage of server side technology that is adverse to WX virtualization. In particular we configured the Web server by relying on JIT (Just-in-Time) compilation for managing interpreted language, like PHP. This can give rise to higher volumes of write and fetch accesses in pages in the address space, hence leading to higher need for kernel level interception of page usage when WX permission is virtualized. The outcome of this study supports anyway the feasibility of WX permission virtualization in such representative server side context. Also, beyond interesting performance results, showing the very limited intrusiveness of this solution, we also report data for assessing its effectiveness in detecting malware, also comparing it with a competitor signature-based malware detector.

Keywords

Crypted malware detection, write-execute permission management, operating system services

1. Introduction

Fighting malware is an increasingly bigger challenge, and several different solutions have been proposed in the literature (see, e.g., [1, 2]). One core aspect that has been addressed is related to encrypted (packed) malware, which typically decrypts itself in executable virtual pages before running the attack steps. This type of malware is particularly complex to face, since techniques like executable-file analysis [3], cannot deal with the runtime installation of malware code on writable/executable virtual pages in the address space when the application is already active. At the same time, some operating system solutions oriented to runtime/dynamic analysis have

ITASEC 2024: The Italian Conference on CyberSecurity, April 08–11, 2024, Salerno, Italy
*Corresponding author.
pasquale.caporsao@cnit.it (P. Caporaso); giuseppe.bianchi@uniroma2.it (G. Bianchi); francesco.quaglia@uniroma2.it (F. Quaglia)
0009-0001-0552-7894 (P. Caporaso); 0000-0001-7277-7423 (G. Bianchi); 0000-0002-5616-7980 (F. Quaglia)

focused their attention on making snapshots and analyzing executable-page contents just when they are accessed for instruction fetch at runtime [4, 5, 6, 7].

Among the recent solutions in this direction, we can find JITScanner [8], a package for security enhancement in the Linux world, in particular with focus on the x86 architecture. The main characteristic of this package, compared to other solutions, is related to its ability to virtualize Write-Execute (WX) permissions on any page of the address space. Hence, it can intercept along time both updates of the page content and any access for fetching instructions from a page with a new content release. This has led JITScanner to be able to identify encrypted malware that is instead not traceable by relying on well diffused solutions oriented to security. In particular, solutions like [9, 10], intercept the initial attempt to fetch an instruction from a modified WX page and mark the page. However, the actual verification of the page content occurs just once, either immediately or at a later time. This could potentially allow an attacker to rewrite the malicious page, concealing its actual content before or after the inspection takes place. JITScanner avoids this problem with its support for the virtualization of WX permission, which allows the system to track any attempt to fetch instructions from a page that hosts any updated content, since its last fetch access.

However, until now JITScanner has been tested in desktop scenarios. Hence, all the experimental results showing its effectiveness—in terms of malware tracking—-and its limited overhead and adequacy to be employed in systems where most of the applications (if not all of them) do not represent malware, are somehow limited.

In this article we provide an experimental study where we test JITScanner in the context of a Web server system. Hence, we enlarge our perspective to the side of services offered in the Internet. In particular, we provide the results of experiments that have been conducted relying on the Apache Web server installed on a Linux release and configured in order to rely on components/modules that can be sees as adversary for the operating mode of JITScanner—in particular concerning the runtime costs it can give rise to. We recall that this mode is essentially based on the management of apposite page-faults that are used to discriminate along time the start of one of the two relevant activities that can happen on a WX page, namely its update or the CPU access for instruction fetch of some fresh content.

In particular, we configured the Web server to use JIT (Just-in-Time) compiling techniques for the management of scripts and programs that can be run by relying on language specific virtual machines, like the PHP runtime system. Hence, the Web server is configured to give rise to an increased volume of both write and execute accesses on virtual pages, which become the target of interceptions by JITScanner. This enabled us to assess the usability and interference of the WX permission virtualization offered by JITScanner in a representative way including 1) adversarial technology (in terms of its effects on the performance we may expect) and 2) interactivity (in terms of latency for the reply to Web requests by specific users/clients).

We believe this experimental study can provide important indications to the community in relation to the feasibility and opportunity to exploit this kind of system level defense solutions.

The remainder of this article is structured as follows. In Section 2, we provide a few basics details on how JITScanner supports the management of WX pages in the address space, which can help better interpreting the experiments we performed for assessing it, and discuss its relation to the literature. The experimental study of the impact of WX permission virtualization, and more generally of the operation mode of JITScanner, on the Apache Web server is presented

in Section 3.

2. Virtualized WX Permission: Concepts and Relation with the Literature

As mentioned, the virtualization of WX permission offered by JITScanner enables the Linux kernel to get control any time a virtual page is accessed for instruction fetch after its content has been updated. This is done with no need for recompiling/reinstalling the Linux kernel, but rather through the usage of Linux Kernel Module (LKM) technology. The solution is therefore simply deployable in already installed versions of Linux.

To precisely identify the moment when a memory page is accessed for an instruction fetch, even after an update of its content, JITScanner relies on the kprobe subsystem provided by the Linux kernel. More precisely, kretprobe has been used for installing a hook on the `handle_mm_fault()` kernel-level procedure. This hook can therefore take a snapshot of the content of the executable page for off-the-critical-path analysis, and can also synchronously check if malware signatures are present, right before the CPU actually goes ahead processing the instructions in that page.

At the same time, the page faults to be intercepted by the hook need to be really triggered. To achieve this objective, the WX permission virtualization approach offered by JITScanner relies on ad-hoc management of both the XD and the W bits on the entries of the page table—we recall that the current implementation of JITScanner is suited for x86 processors. In particular, for each page where WX permission is virtualized, this solution disables both write and execution operations by properly setting the above two bits, hence enabling the raise of a page fault—intercepted by the above mentioned hook—when the update or the fetch of instructions is actually carried out by the CPU. Clearly, just a single one of the two bits can be kept set to enable the corresponding operation at anytime—hence enabling just one of the two possible operations—even though the memory zone the pages belongs too may have been mapped with both write and execute capabilities. Specifically, a single one of the bits is set to enabled when one of the two possible usage-phases (write or fetch) starts, so that the start of the other phase of activity in the page is never missed by the kernel hook (since the other control bit in the page table has been reset, giving rise to the page fault that handles the permission virtualization process).

The last important point is related to the full transparency of these kernel level activities, towards the application level code. In particular, the architecture of JITScanner suppresses the SIGSEGV signal generated by the invalid write/exe access caused by the virtualization of the access permissions to prevent the kernel from terminating the user program or making it run a SIGSEGV handler—in fact, the page fault is generated exactly by the WX virtualization mechanism offered by JITScanner. Hence, the application can write and execute stuff in a page with (virtualized) WX permission according to the conventional Posix specification.

In Linux, faults occurring during instruction fetches are managed within the architecture-specific code of the kernel before calling the `handle_mm_fault()` function. However, these functions cannot be directly hooked using the kprobe mechanism. Hence, JITScanner places a kernel probe on the `force_sig_fault()` function, which is triggered whenever an invalid

Table 1
Comparison with literature solutions

	JITScanner [8]	MAAR [11]	ClamAV [5]	Tracee [12]	Deep-Hook [4]	Falco [13]	Will. et al. [7]
Can be used on non-virtualized environment	✓	✓	✓	✓	X	✓	✓
Allows for dynamic analysis	✓	✓	X	✓	✓	✓	✓
Remains effective against packed malware	✓	✓	X	✓	✓	✓	✓
Uses signatures check	✓	X	✓	X	✓	X	✓
Reduces memory search ranges	✓	na	X	na	X	na	X
Allows monitoring of multiple memory writes on executable pages	✓	na	X	na	X	na	X

memory access occurs.

Overall, this solution enables the identification of the timeline of the interleaves between updates and instruction-fetch on any individual WX page, making it not possible for a malware to run decrypted code that is not analyzed by an external observer.

Additional mechanisms are also added in JITScanner for managing the change of permission in the accesses to virtual pages (e.g. through the `mprotect()` system call), so that the above virtualization scheme for access permissions, and the interception of instruction-fetch from a fresh page content is still carried out even after the (already updated) page is no longer writable.

Given that the focus of this article is on assessing the effectiveness and the intrusiveness of this solution in terms of performance and response latency in the context of services offered on the Web, before entering details of the experimental study we think it is important to provide a comparison of how JITScanner distances from the literature, just thanks to the introduction of WX permission virtualization.

We report in Table 1 the outcome of the comparison considering six alternative solutions. These solutions cover an ample range of options that are currently available for malware detection since they offer either memory checking mechanisms [5, 11, 4, 7] or behavioral analysis [12, 13]. However, even considering advanced solutions like Deep-Hook [4] or the proposal in [7], which essentially offers a step ahead over OmniUnpack [6], the innovation and relevance of JITScanner appears evident. In particular, none of the other solutions enables the check of an executable page multiple times, in particular when a fetch takes place after whichever update of the page. This is an important aspect since the page update is the core block upon which a encrypted malware is built. Also, the intrinsic behavior of JITScanner appears to be well structured since it avoids checks on pages that no thread of the application is actually using for fetching machine instructions. Although this enables amortizing the actual costs for malware detection, we feel the study we provide in the following section can definitely add important hints on the feasibility of JITScanner—and WX permission virtualization—to offer a performance effective solution for continuously monitoring applications and services in the Web, protecting against malware code.

3. Experiments

3.1. Test-bed Platform

All tests have been performed with an Apache2 Web Server running on an Ubuntu Server LTS 22.04 Virtual Machine with Kernel Version 5.15.0 hosted on an Esxi Hypervisor. The machine was equipped with 4 Virtual CPUs and 8 GBs of RAM.

In our study, we augmented JITScanner with new configurations, which allow us to better categorize its performance overhead. The prototype allows for four distinct configurations:

- Virtualization Only: This configuration solely incorporates the functionality of the WX permission shadow state machine. Upon encountering a fault, it intercepts the fault but does not perform any additional actions on the page that triggered it. This configuration serves as a baseline for the other configurations, providing indications only only the overhead associated with context switches between user and kernel modes.
- Transfer Only: In this mode, when a page triggers a fault and transitions to Execution (X) mode, JITScanner copies the page and transfers it to the user agent. Subsequently, the user agent may perform asynchronous analysis to detect potential malicious behavior, in our case this is represented by a simple signature check with YARA rules.
- Sync Only: This configuration implements a synchronous check on the page that triggers the fault, conducted by JITScanner directly within the kernel. Currently, this check involves searching for a single signature throughout the entire page.
- Sync + Transfer: This configuration combines the functionalities of the "Sync Only" and "Transfer Only" configurations described above.

As for the experiments, these can be categorized into two distinct categories: performance and effectiveness. We discuss each category below.

Effectiveness Tests. To assess the effectiveness of JITScanner, we deployed an application known as DVWA (Damn Vulnerable Web Application) on the Web Server. DVWA is a training tool designed to be susceptible to multiple vulnerabilities, in our study we decided to use a Command injection attack. We simulated an attack scenario wherein an adversary attempts to exploit this vulnerability—via specific URLs of the Web server—by loading and executing malware at the server side. Then we installed on the server the ClamAV open-source malware detection tool, as well as JITScanner to evaluate how they respond to such an attack.

For the malware dataset, we selected the "VirusShare_ELF_20200405," which is the latest dataset provided by the free malware sharing website VirusShare, comprising approximately 40,000 samples [14]. To showcase the effectiveness of JITScanner, and of a representative tool we used as competitor, namely ClamAV [5], we executed the samples both in their original state and after packing them with a simple packer sourced online [15]. From the original 40,000 samples we picked 1930 samples that where compatible with our chosen packer and scanned them with ClamAV. Out of these, we extracted 466 samples which ClamAV associated to an immediately identifiable malware family, these are reported in Table 2. After this, we run all

Table 2
Families of the samples used

Family Name	Malware type	Number of samples
Emotet	Trojan	2
Mirai	Botnet	52
Tsunami	Botnet	71
Gafgyt	Trojan	320
XMRIG Miner	Coin-miner	21

samples, both in plain and in packed form, under JITScanner and logged the pages whose accesses in write-execute and/or execute mode have been intercepted[1].

For the logged pages of the samples associated with a family, we chose the most common signature, ensuring that it did not yield any false positives during normal operations in our tests. Finally we added these signatures to the user agent of JITScanner, deploying it and reanalyzing all samples on our automated malware testing facility, namely PHOENIX [16]. Overall, for this effectiveness test, we relied on the Transfer Only setup of JITScanner, thus demanding the malware identification step to the user level agent, while keeping active at kernel level only the WX permission virtualization mechanisms that allows the generation of the snapshot of pages to be checked.

Performance Tests. To quantify the performance overhead of JITScanner on the Web server, we exploited the same Web application utilized in the effectiveness study. This has been done by also configuring the PHP JIT compiler on the Web server to induce additional stress on the system, primary caused by the overhead incurred during the initial execution of instructions fetched from the JIT-compiled pages. Additionally, to further increase the impact of our system on the server, we disabled all forms of caching, including those for the Apache Web server and the PHP engine. This will increase the overhead caused by JITScanner, since it necessitates the re-interpretation of every page upon each request. Successively, utilizing the open-source tool "httperf" we conducted latency measurement on the server. The tool was configured to execute multiple sets of 1000 requests, and we measured the average response time. We opted for this metric in order to focus our analysis on the aspect of interactivity of the client browser vs the Web server.

3.2. Results

Effectiveness Tests. Our testing focuses on two primary objectives:

1. Measuring the "signature flexibility" of JITScanner, which is the extent to which signatures from plain malware remain effective for a variant in the same family.
2. Measuring the "signature retention" of JITScanner, which is the extent to which signatures from plain malware remain effective for their packed counterparts.

[1]All the datasets exploited in this study have been made available at: `https://github.com/Capo80/Malware_Datasets`

Table 3
Signature flexibility of JITScanner and ClamAV

	Family	Detected Plain	Total Samples	Signature Flexibility
JITScanner	Emotet	2	2	**100%**
ClamAV	Emotet	1	2	50%
JITScanner	Tsunami	57	71	**80.2%**
ClamAV	Tsunami	33	71	46.47%
JITScanner	XMRIG_Miner	13	21	**61.90%**
ClamAV	XMRIG_Miner	8	21	38.09%
JITScanner	Gafgyt	76	320	23.75%
ClamAV	Gafgyt	238	320	**74.75%**
JITScanner	Mirai	15	52	**28.84%**
ClamAV	Mirai	8	52	15.38%

To assess point 1, we decided to select the most common signature for each family, which we identified through our log of extracted pages. Subsequently, we compared the effectiveness of this signature against the most common signature detected by ClamAV. Hence, we carried out a fair comparison, considering both ClamAV and JITScanner equipped with somehow equivalent knowledge bases. This is an interesting scenario to test also considering that these two solutions rely on signatures of very different information parts, namely mostly data for ClamAV vs binary code for JITScanner. The results are presented in Table 3, as we can see, JITScanner has a comparable or superior effectiveness across all analyzed families, with the exception of Gafgyt. This discrepancy is likely attributable to the limited number of pages per sample captured in our logs for Gafgyt—the average is 4—which is significantly lower compared to other families, this prevented us from finding an effective signature. We believe that this type of malware detects that it is running on a testing environment (e.g., because of the detection of a reduced number of CPUs) and gives rise to a rapid termination. Such short or failed executions can lead not to intercept actual activities that the malware can (at least potentially) carry out via WX pages that include code signatures—which would have been observed by JITScanner. However, at the same time this is the scenario where the malware becomes essentially idle, not really leading to real security problems.

To assess point 2, we compared the traces extracted from the plain malware to those obtained from the packed samples. It is clear that all signatures are retained if the pages extracted from the plain sample constitute a subset of those extracted from the packed sample. The packed samples will obviously execute over more pages, as they need to decrypt the payload from memory. The details of the experiments are summarized in Table 4. As observed, there exists a substantial difference between the effectiveness of ClamAV and JITScanner when dealing with packed malware. ClamAV lacks a mechanism for detecting signatures at runtime. This makes it not capable to effectively detect packed samples once their actual signature in unpacked in memory for its actual usage. Conversely, JITScanner is able to use the same signatures for almost all samples examined. Also, some malwares are no longer recognized because either the unpacker did not successfully lead to the execution of the malware code (hence JITScanner could not intercept the fetch of the malware instructions from memory) or the unpacking led to different access permissions for a few pages. As for the latter aspect, we found that some

Table 4
Signature retention of JITScanner and ClamAV

	Sign. in Plain	Sign. in Packed	Total Samples	Signature retention
ClamAV	515	0	515	0%
JITScanner	515	391	515	75,91%

Table 5
Slowdown of page requests

Page name	Slowdown
instructions.php	7,5%
php_info.php	7,9%
low.php	3.8%
index.php	3.3%

pages that are setup with read-execute permission with a regular loading of the ELF are instead setup as read-only by the unpacker. In this case, JITScanner does not perform any snapshot of the read-only page for a final check of its content when it is materialized—such snapshot is instead done for read-execute pages, and this gives rise to a different signature characterizing the application.

Performance Tests. The results of the performance tests are shown in Figure 1, and slowdown results—compared to the scenario where the LKM of JITScanner is not mounted at all on the Web server—are reported in Table 5. As for the execution time shown in Figure 1, it has been evaluated by setting up requests on the same machine where the server is hosted, in order to avoid interference and delay contributions related to networking. Hence the reported values represent the server side latency for handling the URL requests. From the data, we can observe that no significant overhead is encountered across any Web server URL, regardless of the size of input/output data involved in the request and the response time. The highest overhead is observed for the most complex page, php_info.php, and reaches approximately 7.9%. The absence of discernible overhead can be attributed to the underlying architecture of PHP JIT, in fact, studies [17] have shown that only a small percentage of code is typically executed in JIT mode on conventional Web pages. Hence, the measured overhead is predominantly caused by the execution and analysis of new pages accessed by server processes, along with the generation of certain WX pages by the PHP JIT compiler itself. This observation is reflected in the negligible overhead incurred by small web pages such as low.php and index.php, which likely contain minimal, if any, code processed through the JIT compiler. Conversely, larger web pages like instructions.php or others which contain more elaborate PHP code, like php_info.php, may undergo some code execution in JIT mode, resulting in a comparatively larger, yet still acceptable, overhead.

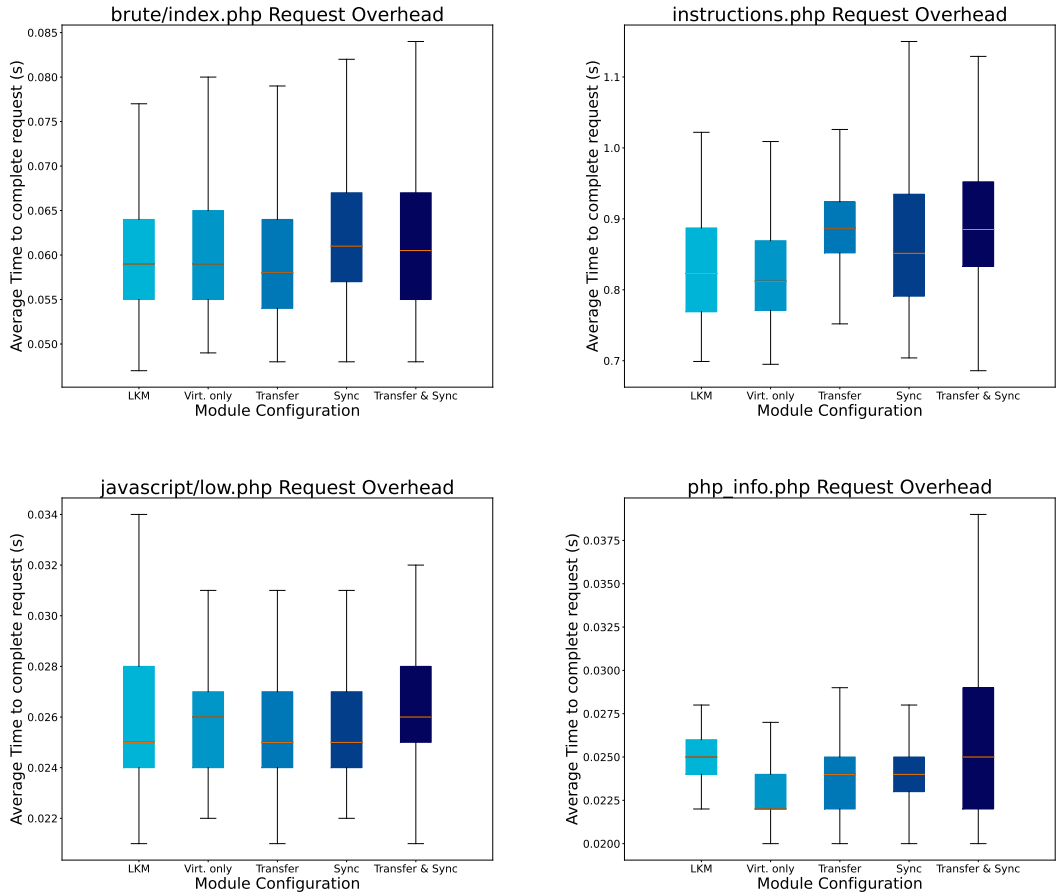

Figure 1: Request Latency under JITScanner

4. Conclusions

System-level engineers are continuously offering new solutions for malware detection. Along this line, one approach that has been recently presented—which works via innovative facilities added to the Linux kernel via LKM technology—is based on supporting the identification of any point in time where a thread executes either the update, or the fetch of instructions from a page with Write (W) and Execute (X) permissions. In other words, each time the CPU fetches an instruction from a page that has been re-updated, the kernel level software takes control, thus working as the hypervisor for the virtualized management of WX permission. The advantage of this solution is the possibility to take snapshots of the (dynamically updated) pages that are actually used for keeping instructions that are executed, giving the possibility to carry out signature checks on the code that a malware software can run at any instant of time—possibly after de-packing it into some WX page.

Till today this solution has been tested in desktop environments, while in this article we

address its benchmarking at the server side. In particular, we tested it in the context of a Web server, configuring the operations of the server side software in a way that is somehow adverse to WX virtualization. More in details, we analyzed the performance that can be ensured via this solution—and hence its overhead—when considering that the Web server is configured to use JIT (Just-in-Time) compiling techniques. This generates larger volumes of occurrences of WX accesses to pages that are destined to host JIT compiled code, which require more frequent interception of the accesses when WX permission is virtualized in the Linux kernel.

Beyond performance, we also tested the effectiveness of the malware detector based on WX permission virtualization, namely JITScanner [8], in such Web server environment, also assessing it against a competitor solution, namely ClamAV [5]. An interesting point in this study is related to the fact that these two solutions are based on signature check on different information portions—mostly data in ClamAV vs binary code in JITScanner. Hence, beyond porting innovation thanks to testing in the server side environment, this article also provides the reader with hints on the outcomes of comparing such two kinds of signature-search methods.

We feel our study can be of real interest for engineers working in the area of security since WX virtualization is a technique that oversteps all previous existing solutions working at kernel level, which rely on the interception of fetches from pages in the address space. In particular, literature studies like [10, 9] can guarantee the signature check of the page content only when it is accessed via fetch for the first time. But they do not support the interception of the fetch from a page anytime it can host a new content—new binary code to be run by a malware, thanks to dynamic updates of its content. WX permission virtualization exactly enables this tracking, hence resulting a valid alternative for security enhancement.

Acknowledgments

This work was partially supported by the project **I-NEST**, "Italian National hub Enabling and Enhancing networked applications & Services for digitally Transforming SMEs and Public Administrations" G.A. 101083398 - CUP F63C22000980006.

References

[1] D. R. Matos, M. L. Pardal, M. Correia, Sanare: Pluggable intrusion recovery for web applications, IEEE Transactions on Dependable and Secure Computing 20 (2023) 590–605. doi:10.1109/TDSC.2021.3139472.

[2] S. Carnà, S. Ferracci, F. Quaglia, A. Pellegrini, Fight hardware with hardware: Systemwide detection and mitigation of side-channel attacks using performance counters, ACM Digital Threats Research and Practice 4 (2023). URL: https://doi.org/10.1145/3519601. doi:10.1145/3519601.

[3] N. Jadvani, M. Agarwal, K. Leelasankar, Malware detection based on portable executable file features, in: A. Ramu, C. Chee Onn, M. Sumithra (Eds.), International Conference on Computing, Communication, Electrical and Biomedical Systems, Springer International Publishing, Cham, 2022, pp. 377–384.

[4] T. Landman, N. Nissim, Deep-hook: A trusted deep learning-based framework for unknown malware detection and classification in linux cloud environments, Neural Networks 144 (2021) 648–685. URL: https://doi.org/10.1016/j.neunet.2021.09.019. doi:10.1016/j.neunet.2021.09.019.
[5] Clamav, https://www.clamav.net/, 2004-2024.
[6] L. Martignoni, M. Christodorescu, S. Jha, Omniunpack: Fast, generic, and safe unpacking of malware, in: Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007), 2007, pp. 431–441. doi:10.1109/ACSAC.2007.15.
[7] C. Willems, F. C. Freiling, T. Holz, Using memory management to detect and extract illegitimate code for malware analysis, in: Proceedings of the 28th Annual Computer Security Applications Conference, ACSAC '12, Association for Computing Machinery, New York, NY, USA, 2012, p. 179–188. URL: https://doi.org/10.1145/2420950.2420979. doi:10.1145/2420950.2420979.
[8] P. Caporaso, G. Bianchi, F. Quaglia, Jitscanner: Just-in-time executable page check in the linux operating system, in: Proceedings of the 18th International Conference on Availability, Reliability and Security, ARES '23, Association for Computing Machinery, New York, NY, USA, 2023. URL: https://doi.org/10.1145/3600160.3605035. doi:10.1145/3600160.3605035.
[9] L. Martignoni, M. Christodorescu, S. Jha, OmniUnpack: Fast, generic, and safe unpacking of malware, in: Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007), IEEE, 2007, pp. 431–441.
[10] C. Willems, F. C. Freiling, T. Holz, Using memory management to detect and extract illegitimate code for malware analysis, in: Proceedings of the 28th Annual Computer Security Applications Conference, ACSAC '12, Association for Computing Machinery, New York, NY, USA, 2012, pp. 179–188.
[11] Z. Salehi, A. Sami, M. Ghiasi, MAAR: robust features to detect malicious activity based on API calls, their arguments and return values, Eng. Appl. Artif. Intell. 59 (2017) 93–102. URL: https://doi.org/10.1016/j.engappai.2016.12.016. doi:10.1016/j.engappai.2016.12.016.
[12] acquasecurity, https://aquasecurity.github.io/tracee/v0.6.4/, 2020–2023.
[13] Sysdig, The falco project, https://falco.org/, 2016–2023.
[14] virus_share, https://virusshare.com/torrents, 2020.
[15] Cyberfined, Cryptor elf loader, https://github.com/cyberfined/cryptor, 2018.
[16] G. Bernardinetti, P. Caporaso, D. Di Cristofaro, F. Quaglia, G. Bianchi, PHOENIX: A cloud-based framework for ensemble malware detection, in: 2023 21st Mediterranean Communication and Computer Networking Conference (MedComNet), IEEE, 2023, pp. 11–14.
[17] php watch, https://php.watch/articles/jit-in-depth, 2020.