CYBERSECURITY OF APPLICATIONS WITH THE MVARMOR TOOL

Łukasz Pietraszek

The Faculty of Electronics and Information Technology, Warsaw University of Technology, Warsaw
corresponding author: lukasz.pietraszek@protonmail.com

Abstract:

This article introduces MvArmor, an MVX system that uses hardware-assisted process virtualization to monitor variants in an efficient but secure way. To provide comprehensive protection against memory error exploits, MvArmor relies on a new variant generation strategy taking into account MVX. The system supports configurable by the user of the security policy in order to reach a trade-off between performance and security. The experiments were divided according to the type of tests performed: safety and performance. The security tests were carried out to verify the correct operation of the MvArmor system, while the performance tests allowed for the measurement of the performance cost caused by the use of MvArmor.

Keywords:
cybersecurity, MvArmor, cyber-attacks, IT

Introduction

In the world of constant technological development and constantly emerging proposals for new solutions, security systems have to face new problems more and more often. One of the most serious application security problems is memory errors in programs. Even the smallest of gaps can compromise users' private data. Over the years, operating systems and special software are trying to implement newer and newer solutions to protect against the use of memory errors, but these safeguards can still be bypassed. The need for defense mechanisms to protect against arbitrary attacks has led to the fight for more comprehensive solutions - primarily Multi-Variant eXecution (MVX) [1].

Multivariate Execution Systems (MVX) increase the effectiveness of software diversity techniques. The key idea is to run multiple and different copies of a program (called variants) in lockstep, providing them with the same input and monitoring their run-time behaviour for discrepancies [1]. Therefore, attackers must simultaneously compromise all variants of the program in order to successfully launch an attack. Many solutions have been proposed, including distributed, heterogeneous MVX systems that use different techniques to further increase the diversity between program variants. However, existing MVX distributed system designs suffer from high overhead costs associated with time-consuming network transactions for multi-variant execution system operations [2, 3].

This article presents the implementation of the MVArmor tool in an experimental environment in order to test the effectiveness of the tool as well as to identify its more detailed advantages and disadvantages. The currently available literature presents a rather modest description of the use of the above-mentioned solutions in information systems. The first proposal for this solution can be found in Koning et al. review from 2016 [1]. Then, that was mentioned in 2019 and 2020 by Liu et al. [4], Dao et al. [5], and Z. Zhang et al. [6]. It should be noted that in each of the above-mentioned cases, the authors described the Multi-Variant execution, but did not focus on a specific tool, and their articles were rather illustrative. The exception is the article from 2016 in which the authors carefully checked the tool's performance on selected servers that may be vulnerable to attacks. In the present article, we would like to focus on MVArmor, checking whether the development of this tool could be useful in the development of cybersecurity in modern IT systems.

Cybersecurity innovation

Cyber Attackers are a group of people who take advantage of people's or systems' mistakes for their own benefit, but admittedly, they have a lot of creativity. Defense against newer and newer attacks forces administrators and users to keep up with new technologies, trends, and their further development. Innovations in cybersecurity result primarily from the need to defend against the creativity of attackers, which in some way enables the development of the entire field [7]. Additionally, due to the Covid-19 pandemic, digitization has become an inseparable part of many areas of social life, and the role of cybersecurity has become even more important than before. The need to accelerate digital transformation, as well as the development of innovative technologies, have influenced the current situation, constantly increasing the number of cyber-attacks [8]. Attackers benefit from every vulnerability, and unfortunately, it seems that this trend will continue to worsen. Nevertheless, it is important to constantly improve security by introducing new security tools.

There are many different sources of information in the field of cybersecurity. One of them is the Survey of Cyber Moving Targets [9], which provides an overview of different cyber moving-target techniques. There is much information about their threat models and technical details. The moving target cyber technique tries to defend the system by making the system more static and less homogeneous. That survey describes the technical details of individual techniques, identifies the appropriate risk model associated with a given technique, and additionally shows its implementation and operation costs. Moreover, the survey describes the weaknesses of individual techniques based on already known attacks and bypassing exploits, but also proposes opportunities for future research in these areas. One of possible solutions is MVArmor [1, 9].

Tools

MvArmor

MvArmor is an MVX system that uses hardware process virtualization to monitor discrepancies between system calls in parallel variants of the program. The use of hardware virtualization enables efficient discrepancy tracking between variants, as it avoids the frequent context switching associated with traditional MVX implementations. In addition, given that the process virtualization layer can

provide MVX with monitoring access to privileged processor functions, the design is particularly susceptible to optimization.

The project was first published at the DSN'16 conference in the article Secure and Efficient Multi-Variant Execution Using Hardware-Assisted Process Virtualization [1]. The original implementation of the solution is available on the Github platform [10]. It consists of a shared library that provides functions for synchronizing system calls, as well as two versions of the module intercepting system calls while the program is running and calling functions from the shared library to synchronize and compare the status of variants. When conducting this research on the MvArmor solution, we relied on its original implementation, using a hardware process virtualization tool called Dune [11].

Environmental Requirements:

- Linux 64-bit x86;
- Intel VT-x virtualization enabled;
- Linux kernel version 3.0 or later.

Dune

Dune is a system that provides applications with direct but secure access to hardware functions, while maintaining existing operating system interfaces for processes. It consists of a small kernel module that initializes the virtualization hardware and mediates interactions with the kernel, and user-level libraries that help applications manage privileged hardware functions [12]. The experiments used Dune for the 64-bit x86 version of Linux.

Ptrace

Ptrace is a Linux mechanism by which a parent process can observe and control the execution of another process. It can inspect and change its core image and registers, and is mainly used to implement breakpoint debugging and system call tracing [13]. Ptrace was used in the original MvArmor implementation as a simpler and less efficient alternative to the Dune utility, mainly used for debugging purposes.

Httpd

Daemon HTTP is software that runs in the background of a web server and waits for incoming server requests. Daemon automatically responds to the request and handles hypertext and multimedia documents over the Internet using the HTTP protocol [14]. The CGI protocol works with it, which adds scripts with the request based on which the content is delivered instead of the static content being returned. The CGI protocol is not necessary, but HTTPd and CGI are used together to deliver dynamic content [15, 16]. During the experiments, an implementation of the HTTP server from the Apache project (httpd) was used.

Experiments

Experimental environment

Experiments were performed in the Linux Ubuntu 14.04.1 64-bit version with the use of components:

- CPU: Intel i7-5600U 2,6GHz;
- RAM: 4GB.

Implementation

Koning, Bos, & Giuffrida [1] created C libraries with approximately 5000 lines of code. Having access to the address space of the monitored application, the software is able to determine how memory is allocated between the variants, as well as monitor the state of the application memory in order to detect a potential attack. The project is split into two implementations: Sandbox Dune, which allows running and monitors any application, and ptrace, which is used for development and debugging. In order to test the effectiveness of the solution, the authors used then popular server programs that could be exposed to remote attacks. They have chosen nginx (v0.8.54), lighttpd (v1.4.28), bind (v9.9.3), and beanstalkd (v1.10) for their experiments.

Security experiments

In order to verify the effectiveness of MvArmor in terms of detecting and preventing attacks using memory errors, a simple C program was written containing a vulnerability allowing for a buffer overflow attack. The program contained several lines of code in which a string of arbitrary length, passed as an argument of the program call, was copied to a fixed-length buffer.

To enable a successful buffer overflow attack, the program was compiled with gcc compiler safeguards disabled, such as disabling stack execution, and detecting and terminating stack overflow attempts. The ASLR (Address Space Layout Randomization) mechanism built into the Linux system, consisting in the randomization of memory addresses allocated to processes (which is a separate technique of the Moving Target Defense type), was also deactivated.

The attack consisted in overwriting the return address from the function with the address pointing to the area belonging to the program stack, where the code running the program /bin/dash was placed. Attempts were made to attack the version of the program without MvArmor protection, as well as the version launched using MvArmor.

In the case of the program version without additional protection, the introduction of malicious data resulted in unexpected behaviour of the program and the launch of the /bin/dash program. However, the program with the MvArmor protection working was terminated before the stack contents were executed, and the MvArmor logs could read information about a discrepancy in the content of system calls, indicating an attack. Therefore, it was possible to confirm the ability of the MvArmor solution to detect and prevent simple buffer overflow attacks based on overwriting the return address from the function.

Performance experiments

In order to measure the performance cost of using MvArmor, the HTTP server from the Apache project was launched and its response latency was measured as well as the number of completed

requests per unit of time. Experiments were performed for the server version without MvArmor protection and with a different number of parallel variants launched by the MvArmor utility. The autocannon tool [17] was used to perform the experiments. It allowed to set up many parallel connections with a given server and send requests to it for a specified number of seconds.

The conducted experiments investigate the impact of the increase in the number of parallel connections on selected configurations, such as the average delay, the average number of requests, and the number of errors returned by the HTTP server. The experiments were performed 4 times and the results were averaged in order to obtain the most optimal results.

Tab. 1. Average delay

Number of parallel connections	Without MVArmor	MVArmor with 2 variants	MVArmor with 3 variants	MVArmor with 4 variants
10^0	8488ms	7971ms	7501ms	7259ms
10^1	7776ms	8051ms	7498ms	7089ms
10^2	6901ms	7151ms	7110ms	7168ms
10^3	6612ms	7120ms	3861ms	2871ms

Source: own calculations

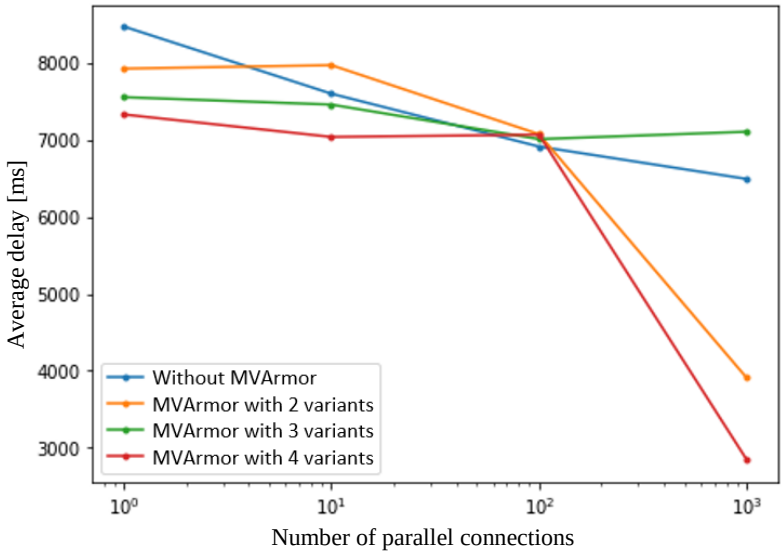

Fig. 1. Average delay
Source: own calculations

Fig. 1. shows the average values of response delays with a different number of parallel connections to the server, for different versions of the server (without MvArmor protection and with MvArmor protection, with a different number of variants running).

Average latency drops for MvArmor with 2 and 4 variants. This is because the increase in the number of parallel connections makes the system unable to handle them. In the case of experiments for a system without MvArmor defence and for MvArmor with two variants, a general downward trend in the average delay can also be seen, but it is not as marked a decrease as in the case of the 2 and 4 variant configurations.

Interestingly, the latency for the version with MvArmor protection is often lower than for the unprotected server.

Tab. 2. Average number of requests

Number of parallel connections	Without MVArmor	MVArmor with 2 variants	MVArmor with 3 variants	MVArmor with 4 variants
10^0	69	92	31	28
10^1	258	236	227	201
10^2	239	238	261	238
10^3	315	240	260	209

Source: own calculations

Fig. 2 shows the average number of completed requests for the same configuration options. The average number of completed requests is generally lower for the version of the server with MvArmor protection than for the version without protection. However, these differences are not big. Again, these are not big differences, and it can be explained to some extent by the lower number of requests for servers with MvArmor protection. Average number of requests for 10-1000 parallel connections varies between 201 and 261 for all server version.

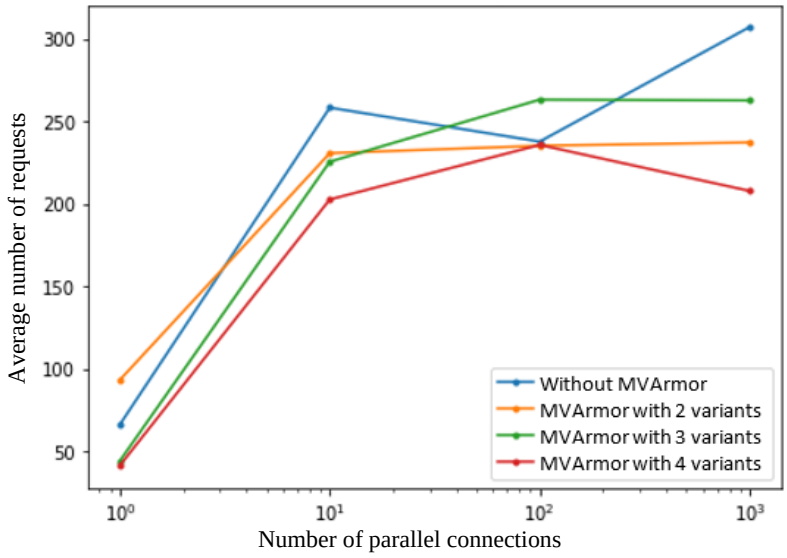

Fig. 2. Average number of requests
Source: own calculations

Tab. 3. The number of errors returned by the HTTP server

Number of parallel connections	Without MVArmor	MVArmor with 2 variants	MVArmor with 3 variants	MVArmor with 4 variants
10^0	0	0	0	0
10^1	0	0	0	0
10^2	2	0	0	0
10^3	548	1320	464	1602

Source: own calculations

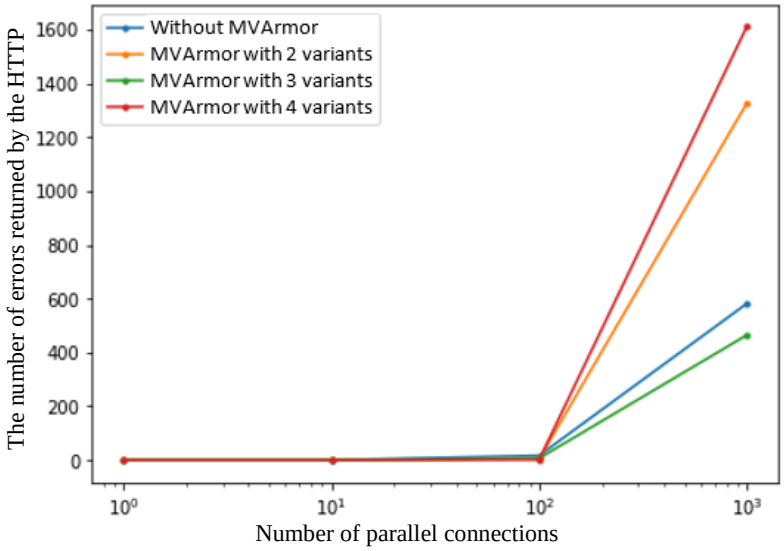

Fig. 3. The number of errors returned by the HTTP server
Source: own calculations

Fig. 3 shows the number of errors returned from the server in response to a request for different configurations. The errors returned by the HTTP server do not appear until 1000 concurrent connections. This coincides with the average delay, which for the same number of parallel connections suddenly drops drastically for a MvArmor configuration with 2 and 4 variants.

During the experiments, a performance study was also conducted based on CPU and RAM utilization. For each type of experiment, RAM consumption remained around 20-30% all the time. In the case of the processor, consumption was almost always below 10%, small jumps were not related to the experiments performed. The exception was the test of the number of errors returned by the HTTP server. For 10^3 parallel connections, the CPU was above 70%, and on one attempt, the system became unresponsive.

Discussion

The presented solution has some limitations, mainly hardware. According to the authors [1], Linux kernel version 3.0 or newer is required to run the tool, however, experiments show that the solution cannot be built with kernel version 4.0 or newer.

The use of implementation used requires a very specific version of the operating system. The authors [1] indicate that the solution was successfully evaluated on Ubuntu version 14.04. The Linux kernel version requirements are also met by Ubuntu 14.10, however, it was not possible to build and run the solution on this version of the system due to bugs and missing libraries. Problems with running MvArmor also occurred on Ubuntu version 14.04.06 (LTS); finally, the solution was built and run on Ubuntu version 14.04.01. The issues listed are likely due to specific Dune requirements.

An important limitation is also the fact that Dune works only for processors from the Intel family.

Conclusions

MvArmor is an effective solution, but it generates many compatibility problems. The solution has been tested on many different versions of the Linux Ubuntu operating system, however, the only version that has successfully run the presented implementation is Linux Ubuntu 14.04.01 64-bit x86.

The tests performed did not reveal a clear performance overhead caused by the use of MvArmor. However, a more accurate assessment of the cost of using the MvArmor solution requires further research.

Literature

[1] K. Koning, H. Bos, C. Giuffrida, *Secure and efficient multi-variant execution using hardware-assisted process virtualization,* 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), (2016), 431-442.

[2] S. Volckaert, B. Coppens, B. De Sutter, K. De Bosschere, P. Larsen, M. Franz, *Taming parallelism in a multi-variant execution environment*, Proceedings of the Twelfth European Conference on Computer Systems, (2017), 270-285.

[3] S. Österlund, K. Koning, P. Olivier, A. Barbalace, H. Bos, C. Giuffrida, *kMVX: Detecting kernel information leaks with multi-variant execution*, Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, (2019), 559-572.

[4] Z. Liu, Z. Zhang, J. Zhang, H. Liu, *Multi-Variant Execution Research of Software Diversity*, Journal of Physics: Conference Series, (2019), Vol. 1325(1), 012107.

[5] D. Yao, Z. Zhang, G. Zhang, *Practical Control Flow Integrity using Multi-Variant execution*, Proceedings of the 2020 International Conference on Internet Computing for Science and Engineering, (2020), 14-19.

[6] G. Zhang, Z. Zhang, B. Ma, J. Wang, *Multi-variant execution: State-of-the-art and research challenges*, 12th International Conference on Communication Software and Networks (ICCSN), (2020), 196-201.

[7] S. Petrenko, *Cyber security innovation for the digital economy*, Innopolis, Russia: River Publishers 2018.

[8] B. Pranggono, A. Arabo, *COVID-19 pandemic cybersecurity issues*. Internet Technology Letters, (2021), Vol. 4(2), e247.

[9] B. C. Ward, S. R. Gomez, R. Skowyra, D. Bigelow, J. Martin, J. Landry, H. Okhravi, *Survey of cyber moving targets second edition*, Lexington, Massachusetts: MIT Lincoln Laboratory2018.

[10] *Github MvArmor*, https://github.com/vusec/mvarmor, 13.06.2022.

[11] A. Voulimeneas, D. Song, P. Larsen, M. Franz, S. Volckaert, *dMVX: secure and efficient multi-variant execution in a distributed setting*. Proceedings of the 14th European Workshop on Systems Security, (2021), 41-47.

[12] A. Belay, A. Bittau, A. Mashtizadeh, D. Terei, D. Mazières, C. Kozyrakis, *Dune: Safe user-level access to privileged {CPU} features*, 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12), (2012), 335-348.

[13] J. Keniston, A. Mavinakayanahalli, P. Panchamukhi, V. Prasad, *Ptrace, utrace, uprobes: Lightweight, dynamic tracing of user apps*, Proceedings of the 2007 Linux symposium, (2007), 215-224.

[14] R. T. Fielding, *Shared leadership in the Apache project.* Communications of the ACM, (1999), Vol. 42(4), 42-43.

[15] D. Q. Naiman, *Statistical anomaly detection via httpd data analysis.* Computational statistics & data analysis, (2004), Vol. 45(1), 51-67.

[16] P. D. Pandit, *Implementation of HTTP Server in Linux*, https://www.researchgate.net/publication/354935380_Implementation_of_HTTP_Server_in_Linux, 20.06.2022.

[17] *Github autocannon*, https://github.com/mcollina/autocannon, 13.06.2022.