A Multi-variant Execution Environment for In-memory Databases

Shuhei Enomoto (Student)
TUAT
Tokyo, Japan
enomoto@asg.cs.tuat.ac.jp

Hiroshi Yamada
TUAT
Tokyo, Japan
hiroshiy@cc.tuat.ac.jp

The multi-variant execution environment (MVEE) is a useful approach to enhancing security of system software written in unsafe languages such as C and C++. Although a number of security mechanisms, e.g. Address Space Layout Randomization (ASLR) and Address/Undefined Behavior Sanitizers (ASan and UBSan), are supported at the kernel and compiler level, each mechanism defends only a specific attack and multiple mechanisms cannot be used at a time due to their implementation conflicts. The MVEE allows us to concurrently run security mechanisms by executing several variants of the target software, integrating a security mechanism into a variant, and synchronizing and monitoring their behavior against the same inputs from disks and networks. The previous researches have shown its effectiveness for user-level applications [5] and OS kernels [4].

Applying the MVEE to in-memory databases (DBs), which are a key component in modern web systems and suffer from memory vulnerabilities [1, 2], poses a new challenge stemming from tremendous memory space overhead. Since in-memory DBs, e.g. memcached and redis, manage data items and the running states on their own memory region, the memory footprint is much larger than the legacy stateless applications such as web servers. For example, when the MVEE executes an in-memory DB with 512 GiB of memory that is an instance offered in real-world cloud systems [3], we require more than 1 TB of memory in a case where three variants are launched. This makes it hard or sometimes impossible to use the MVEE for in-memory DBs and causes inefficient resource utilization in cloud systems to perform less service consolidation.

We present an MVEE mechanism to execute multiple variants of in-memory DBs in an memory-efficient manner. The key behind our approach is that the memory contents of in-memory DB's variants are quite similar to each other; all the variants place the *same* data items in their memory heaps and most of their memory regions is occupied by the data items. To reduce the memory usage caused by multiple variant launches, our approach shares the same contents memory among variants while simultaneously monitoring behavior of different security-enhanced variants.

Our MVEE is driven by the following design goals; (1) enhances security as the same as the existing MVEE, (2) restricts the total memory utilization of the MVEE as much as possible, and (3) no modification of the application source code/binary. To satisfy these goals, our mechanism leverages a kernel-level page sharing feature that merges the same contents pages into one page transparently to running applications and shares the page in a copy-on-write manner. Our MVEE, running on the kernel, monitors and synchronizes each variant at the system call level and executes the page sharing mechanism per a certain interval. The MVEE also hooks lock/unlock functions to support multi-threaded applications. It forces running threads in each variant to acquire locks in the same order to equalize the system call sequences among the variants.

We conducted preliminaly experiments about memory usage. Our prototype runs on Linux 4.4.185 and successfully executes two real-world in-memory DBs, memcached and Redis, with ASan, UBSan and ASLR. The experimention using the memtier_benchmark shows that the our prototype successfully reduce memory usages by up 56.9 % smaller than the vanilla MVEE.

This work is ongoing. Our current focus is on a performance issue of our mechanism. The runtime overhead stemming from page sharing is 34.3 %. This is because the prototype reuses the existing page sharing feature of Linux and thus there is a room for optimization. We are now designing a new page sharing feature that frequent updated pages are not merged to avoid page merging and copy-on-write overhead.

References

[1] CVE Details. 2019. Vulnerability Details : CVE-2019-10193. https://www.cvedetails.com/cve/CVE-2019-10193/.

[2] CVE Details. 2019. Vulnerability Details : CVE-2019-15026. https://www.cvedetails.com/cve/CVE-2019-15026/.

[3] Amazon Web Services Inc. 2012. Amazon DynamoDB Accelerator (DAX). https://aws.amazon.com/dynamodb/dax/.

[4] Sebastian Osterlund, Koen Koning, Pierre Olivier, Antonio Barbalace, Herbert Bos, and Cristiano Giuffrida. 2019. kMVX: Detecting Kernel Information Leaks with Multi-variant Execution. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'19)*. Providence, RI, USA, 559–572.

[5] Meng Xu, Kangjie Lu, Taesoo Kim, and Wenke Lee. 2017. Bunshin: Compositing Security Mechanisms through Diversification. In *Proceedings of the 2017 USENIX Annual Technical Conference (ATC'17)*. USENIX, Santa Clara, CA, 271–283.