Performance Improvement of Dynamic Binary Analysis Tool, Triton

Won-Il Pyo, Beom-Su Hyeon, Ye-Byoul Son, and Mun-Kyu Lee

Abstract—Dynamic binary analysis (DBA) programs are used to automatically find vulnerabilities in programs. Among them is Triton, a recently developed open-source DBA program with user-friendly functionalities such as Python interface. In this paper, we improve the performance of Triton by applying a caching technique to the opcode disassembly task in Triton. We designed and implemented two modified version of Triton and compared their performance with the original version. According to our experimental results, the speed of Triton was accelerated by up to 25% on average. This improvement was due to the high hit ratio of the caches we added in our modification, which was higher than 95%.

Research Keywords—Dynamic binary analysis (DBA), Space-time trade-off analysis, Triton

1 Introduction

There are various dynamic binary analysis (DBA) programs, e.g., Valgrind [1], QEMU [2], DynamoRIO [3], Triton [4], etc., which analyze programs dynamically when they are running. These DBA tools can also be used to find vulnerabilities in a program automatically, which can help programmers escape from hard the boring manual anlaysis tasks. However, most of these DBA programs are very slow due to the significant overhead for performing analysis tasks and executing the original program synchronously.

In this paper, we modify one of these DBA program, Triton, and improve its performance. Triton is a C++ based open-source DBA framework whose development was started on January 2015 at the Bordeaux University [4, 5]. It has many user-friendly functionalities such as Python interface, and is at active development stage. It is expected that Triton would be used for various DBA applications We improve the performance of Triton by applying a caching technique to the opcode disassembly task in Triton.

- *Won-Il Pyo is with the Department of Computer Engineering, Inha University, South Korea. E-mail: pyowonil@gmail.com.*
- *Beom-Su Hyeon is with the Department of Computer Engineering, Inha University, South Korea. E-mail: ksitht@gmail.com.*
- *Ye-Byoul Son is with the Department of Computer Engineering, Inha University, South Korea. E-mail: byoul0114@gmail.com.*
- *Mun-Kyu Lee (corresponding author) is with the Department of Computer Engineering, Inha University, South Korea. E-mail: mklee@inha.ac.kr.*

2 Proposed Method

Triton can dynamically analyze a target program in cooperation with dynamic binary instrumentation (DBI) tools. The final goal of Triton is to support various DBI tools, but currently it only supports Pin, Intel's DBI tool for IA-32 and IA-64 [6]. Pin executes binary instrumentation codes, inserting DBA codes provided by Triton. For this purpose, opcodes are fetched, disassembled and transformed to an instance of the Instruction class defined by Trion. The disassembly procedure is performed by Capstone, a disassembly framework [7].

The motivation for our work is that the same disassembly procedure is repeated for the same opcode. Our idea for performance improvement is to use the principle for cache memory, i.e., space-time trade-off. According to this principle, we designed and implemented software cache memory. After an opcode is disassembled, it is stored in this cache. Later, if the Instruction object of an opcode to be disassembled is already in cache, then we can reuse this object without disassembly.

To implement the proposed software cache, we used the map container provided by the C++ standard template library (STL). It is well known that the map class guarantees efficient search operations because it is implemented with a red-black tree whose worse-case time-complexity for a search operation is O(log n), where n is the number of entries in the map. Note that the most dominant operation for our situation is a search operation because a search operation is necessary for both cache hit and miss.

In general, each data entry stored in a map is accessed using its unique key. We used opcodes as unique keys to store data entries, i.e., Instruction objects corresponding to those opcodes. However, the length of an opcode used for Triton is 256 bits long, which is too long for a key. Therefore, we also considered a variant where a 32-bit hash value of an opcode is used as a key. For convenience we call the former and the latter 'T256' and 'T32,' respectively. To distinguish the unmodified Triton from the above modified versions, we call it 'T.'

3 Implementation and Experiment

We implemented the proposed methods and verified through experiments the performance improvement in Triton with the proposed method. Te be precise, we compare the performance of T, T32, and T256. Table 1 shows the setup for our experiments.

For user's convenience, Triton provides a Python API which can be accessed through Python scripts. These scripts specify how the target program should be analyzed. For fair comparison, we used the same script for T, T32, and T256.

To generate 32-bit keys for T32, we used the boost hash algorithm [8]. This hash algorithm takes a 256-bit opcode as input and produces a 32-bit hash value. Note that a hash algorithm has a collision problem, and we must resolve collisions. In our case, two 256-bit inputs may be mapped to the same 32-bit hash result. This means that two distinct opcodes can share the same hash value in T32. That is, even when a 32-bit key for the current opcode is found in the cache, the corresponding Instruction object retrieved from the cache with this key may be different from what should have been the result of disassembly of the current opcode. Our solution to resolve this issue is to use another cache. This additional cache stores the original 256-bit opcode using its 32-bit hash value as the key. At the moment when an opcode should be disassembled, we first look into this second cache and check if there is already a data entry with the same key value. If it is the case, the 256-bit data entry in the cache is compared with the current opcode. If they are the same, this is the cache hit, and the Instruction object with the same 32-bit key value is retrieved from the first cache. On the other hand, the following two cases are cache misses, and a disassembly operation is performed: (1) there is no date entry with the 32-bit key in the second cache or (2) there is a data entry with that key in the second cache, but the 256-bit opcode is different from the current one. Note that the second cache is not required for T256 because it does not use any hash. Table 2 summarizes the cache setup for T, T32, and T256.

We measured the running time of Triton with Linux time command. We considered three simple shell commands, pwd, id, and ls and separately measured the timings for three versions of Triton using T, T32, and T256, respectively. The results are shown in Table 3 and 4.

Table 1. Experimental setup

OS	Ubuntu 16.04 LTS, 4.4.0-57-generic (64-bit)
CPU	Intel(R) Core™ i7 CPU 870 @ 2.93GHz (8-core)
Memory	8GB RAM / 128GB SSD

Table 2. Cache setup

	T	T256	T32
Number of cache	0	1	2
Maximum cache size	-	$O(2^{256})$	$2O(2^{32})$

Table 3. Measured time (run-time and speedup)

Target Program	T	T256	T32
pwd	15.60s	12.27s;127%	12.11s;129%
id	28.04s	22.72s;123%	22.49s;125%
ls	51.31s	41.07s;125%	40.82s;126%

Table 4. Measured hit ratio (%)

Target Program	T	T256	T32
pwd	-	97.33	97.12
id	-	95.70	95.39
ls	-	96.63	96.32

4 Conclusions

In this paper, we improved the performance of Triton, one of the recently developed DBA programs. We designed and implemented two modified versions of Triton using caching techniques and compared their performance with the original version. According to our experimental results, the speed of Triton was accelerated by up to 25% on average.

Acknowledgment

This paper was presented at platcon-17 (2017 In-ternational Conference on Platform Technology and Service) with the same title. This work was supported in part by the MSIT (Ministry of Science and ICT), Korea, under the ITRC (Information Technology Research Center)

support program (IITP-2017-2012-0-00646) supervised by the IITP (Institute for Information & communications Technology Promotion) and by the IITP grant funded by the Korea government (MSIT) [No.R7122-16-0077, Development of the 10G Level RegEx Packet Pro-cessing SW for the Security Intelligence].

REFERENCES

[1] http://valgrind.org/
[2] http://wiki.qemu.org/Main_Page
[3] http://www.dynamorio.org/
[4] https://triton.quarkslab.com/about
[5] https://github.com/JonathanSalwan/Triton
[6] https://software.intel.com/sites/landingpage/pintool/docs/49306/Pin/html/
[7] https://www.capstone-engine.org/
[8] http://www.boost.org/doc/libs/1_35_0/doc/html/boost/hash_range_id420926.html

Won-Il Pyo Received the B.S. degree in Computer Science and Engineering from Inha University in 2016. He is currently a student in the M.S. course in the Department of Computer Engineering at Inha University. His research interests are in the areas of information security and post-quantum cryptography.

Beom-Su Hyeon He is currently a student in the B.S. course in the Department of Computer Engineering at Inha University. His research interests are in the areas of artificial intelligence and binary exploitation.

Ye-Byoul Son She is currently a student in the B.S. course in the Department of Computer Engineering at Inha University. Her research interests are in the areas of network security and machine learning.

Mun-Kyu Lee Received his B.S. and M.S. degrees in Computer Engineering from Seoul National University in 1996 and 1998, respectively, and his Ph.D. degree in Electrical Engineering and Computer Science from Seoul National University in 2003. From 2003 to 2005, he was a senior engineer at Electronics and Telecommunications Research Institute, Korea. He is currently a professor in the Department of Computer Engineering at Inha University, Korea. His research interests include information security and theory of computation.