Compatibility of trusted computing

Qi Hu

01/11/2022

Keywords: Trusted Computing, Distribute System, Software-hardware Co-optimization.

1 Introduction

2 Research Objectives

1. Design a dynamic analysis/translation tool to support running legacy software on trusted computing systems. This tool can be implement into 3 parts:
   - Dynamic analysis of legacy software, point out which parts of the code are more vulnerable to attack and which data are important and need to be protected.
   - Using the analyzed data to guide the in-place binary translation, insert new code segments, which are used to protect code and migrate data into enclave.
   - Binary translation is used for cross-instruction architecture programs. The migration instruction segments can be generated and added during the translation process.
2. Migrate the whole dynamic analysis/translation tool into a distribute system.
3. Explore performance bottlenecks and help design/modify the hardware architecture. Achieve efficient operation of the entire system with the help of hardware-software co-optimization.

3 Background of Research

3.1 Trusted Computing and TEE

Various encryption and authentication methods (e.g. TLS and file disk encryption) are often used to prevent confidential data loss, theft or corruption. However, relying solely on software for confidential data protection has many problems, such as software vulnerabilities, reverse engineering cracking [25], etc. So, it is useful to use Trusted Execution Environment(TEE) to protect encryption software and data, which provides an environment shielded from outside interference and the necessary mechanisms to build secure and sensitive applications.

Intel Software Guard Extensions(SGX) is a set of security architecture extensions [12]. It provides the enclave environment which can prevent all other software accessing the code and data located inside an enclave. Also when data leaves the enclave and written into the memory, the data will be automatically encrypted.

ARM TrustZone uses a different approach to TEE by introducing a secure world, which is a new execution environment in the processor in addition to the normal world [13]. The secure world has

multiple privilege levels just like an virtual machine(VM), which provides the opportunity to implement an entire trusted software stack.

Due to overly complex operations and unacceptable hardware overhead, Intel begin moving to Trust Domain Extensions(TDX) [18, 19], a new trusted computing architecture introduces a separate trusted hypervisor/VMM. The interaction between trusted virtual machines and external untrusted environments need to be checked by the security check module Shim.

Since TrustZone lacks confidentiality support, ARM v9 proposes Confidential Compute Architecture (CCA). CCA differs from TrustZone, which supports in-memory confidentiality capabilities directly in hardware, specifically designed to protect users' confidential data [3].

3.2 Binary rewriting and binary translation

Binary rewriting is a technique for modifing or translating the original binary code without having the source code. According to their characteristics, they can be divided into four categories: static, dynamic, minimal-invasive and full-translation.

Static binary rewriting can use the existing information, such as static data flow analysis and symbol table information, to optimize or enhance existing programs [22, 21]. Dynamic binary rewriting is performing alterations during execution, which can be used for performance analysis [11] and hot code patching [5]. Minimal-invasive rewriting is based on branch granularity. It will perform additional instruction at the original location by rewriting to branch instructions. This is often used to add new function to the original program [6]. Full-translation rewriting can convert binaries at any instruction, and usually lift origin binary code into intermediate representations for translating. Some open source tools, like QEMU [4] and Valgrind [15], use full-translation for binary rewriting.

3.3 Distribute system in Trusted computing

With the rise of cloud computing and increase in data sets in recent years, more and more scenarios require the use of distribute systems. While distribute systems, such as Hadoop and Spark, are receiving an increasing number of threats.

In 2015, the first distribute MapReduce system VC3 was proposed, which keeps the code and data confidential, ensures the correctness and completeness of the results [20]. SGX-PySpark was implemented in 2019, and with the help of TEE, it can protect the confidential data [17].

For other systems, such as database, EnclaveDB uses SGX to protect the database engine and ensure high performance [16]. EncDBDB also uses SGX for data security and is optimized for column-oriented in-memory databases [7].

In recent years, heterogeneous computing systems such as Computation Storage Architectures (CSA) has also faced data security issues. IronSafe provides a secure processing system for heterogeneous computing storage architectures using a hardware-assisted trusted execution environment [24].

4 Research Plan and Methodology

4.1 Design a tool to analyze legacy programs

In order to complete the migration of a legacy system, the first step is to analyze the program and identify the code and data that needs to be migrated. There are three parts we need to consider: data analysis, recognition algorithm and recognition accuracy.

The first is **data analysis**. The purpose of the analysis is to provide information about binaries so that the subsequent identification process can proceed smoothly. There are many similar works in the

field of binary rewriting to explore the accuracy of binary analysis. BIRD [14] uses a combination of static and dynamic identification methods to improve the accuracy of the analysis. However, for dynamic disassembly, some trapped instruction need to be inserted which is not a good choice for us. Other papers point out that static disassembly can also achieve good results [2]. So we prepare to implement the static analysis tool first and then decide whether to add dynamic analysis based on the accuracy of the analysis.

The next is the **recognition algorithm**. Moat [23] is a detection tool designed by Berkeley that uses automatic theorem proving and information flow analysis methods to discover the possibility of applications leakage of secret information in the SGX region by analyzing the assembly language level of the program. Our work can be based on Moat, from which we can extract effective identification and verification algorithms and use them in our analysis tools.

In order to evaluate **recognition accuracy**, we will consider it in two parts. *Coverage of analysis*: We will use different test cases to see how well the overall analysis is covered. Since we have access to the source code of these test programs, the coverage of the analysis can be measured by some tools such as gcov [8] and QEMU [4]. *Correctness of the analysis*: We also use the test cases available in the source code to verify correctness. We would like to compare the results of our analysis tool with the results of the source code after automatic analysis by Glamdring [9] in order to obtain an accurate analysis.

4.2 Design a binary rewriting tool to protect confidential code and data

With the help of analysis tools, it is easy to obtain the code segments that need to be protected and the memory areas where important data is located.

Code protection. For code segments that need to be protected, we can use the Minimal-invasive translation method. Similar to the rev.ng [6] and pin [11], we can insert the required functions before and after the code segments. We insert the enclave's entry code and enclave's parameters passing code before the segments, also enclave's return parameters can be built at the end of the segments. There should be many more details to note and consider here that need to be discovered and resolved in the course of research.

Data protection. Data protection is more difficult than code protection, especially for global variables. For local variables, we can analyze them, get the program boundary, and put the variables as well as code into enclave for protection. PtrSplit focuses on C/C++ pointers, identifies pointers that block the generation of partition boundaries [10]. But for global variables, there is no good solution for now. However, for a highly cohesive and low-coupling system, global variables are often not recommended [1], so dropping this part of the protection when it cannot be solved is generally not a big deal.

4.3 Extend the binary rewriting tool into distribute system

4.4 Optimize the above system by software-hardware co-optimization

5 Expected Outcomes and Significance

6 Study Schedule

References

[1] Global variables are bad. http://wiki.c2.com/?HeuristicRule.

[2] Andriesse, D., Chen, X., van der Veen, V., Slowinska, A., and Bos, H. An in-depth analysis of disassembly on full-scale x86/x64 binaries. In *USENIX Security Symposium* (2016).

[3] ARM. Arm confidential compute architecture. https://www.arm.com/en/architecture/security-features/arm-confidential-compute-architecture.

[4] Bellard, F. Qemu, a fast and portable dynamic translator. In *USENIX Annual Technical Conference, FREENIX Track* (2005).

[5] Bruening, D., Garnett, T., and Amarasinghe, S. P. An infrastructure for adaptive dynamic optimization. *International Symposium on Code Generation and Optimization, 2003. CGO 2003.* (2003), 265–275.

[6] Federico, A. D., Payer, M., and Agosta, G. rev.ng: a unified binary analysis framework to recover cfgs and function boundaries. *Proceedings of the 26th International Conference on Compiler Construction* (2017).

[7] Fuhry, B., JayanthJainH, A., and Kerschbaum, F. Encdbdb: Searchable encrypted, fast, compressed, in-memory database using enclaves. *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)* (2021), 438–450.

[8] GNU. gcov – a test coverage program. https://gcc.gnu.org/onlinedocs/gcc/Gcov.html.

[9] Lind, J., Priebe, C., Muthukumaran, D., O'Keeffe, D., Aublin, P.-L., Kelbert, F., Reiher, T., Goltzsche, D., Eyers, D., Kapitza, R., Fetzer, C., and Pietzuch, P. R. Glamdring: Automatic application partitioning for intel sgx. In *USENIX Annual Technical Conference* (2017).

[10] Liu, S., Tan, G., and Jaeger, T. Ptrsplit: Supporting general pointers in automatic program partitioning. *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (2017).

[11] Luk, C. K., Cohn, R. S., Muth, R., Patil, H., Klauser, A., Lowney, P. G., Wallace, S., Reddi, V. J., and Hazelwood, K. M. Pin: building customized program analysis tools with dynamic instrumentation. In *PLDI '05* (2005).

[12] McKeen, F. X., Alexandrovich, I., Berenzon, A., Rozas, C. V., Shafi, H., Shanbhogue, V., and Savagaonkar, U. R. Innovative instructions and software model for isolated execution. In *HASP '13* (2013).

[13] Mukhtar, M. A., Bhatti, M. K., and Gogniat, G. Architectures for security: A comparative analysis of hardware security features in intel sgx and arm trustzone. *2019 2nd International Conference on Communication, Computing and Digital systems (C-CODE)* (2019), 299–304.

[14] Nanda, S., Li, W., Lam, L.-C., and cker Chiueh, T. Bird: binary interpretation using runtime disassembly. *International Symposium on Code Generation and Optimization (CGO'06)* (2006), 12 pp.–370.

[15] Nethercote, N., and Seward, J. Valgrind: a framework for heavyweight dynamic binary instrumentation. In *PLDI '07* (2007).

[16] Priebe, C., Vaswani, K., and Costa, M. Enclavedb: A secure database using sgx. *2018 IEEE Symposium on Security and Privacy (SP)* (2018), 264–278.

[17] Quoc, D. L., Gregor, F., Singh, J., and Fetzer, C. Sgx-pyspark: Secure distributed data analytics. *The World Wide Web Conference* (2019).

[18] Sahita, R., Caspi, D., Huntley, B. E., Scarlata, V., Chaikin, B., Chhabra, S., Aharon, A., and Ouziel, I. Security analysis of confidential-compute instruction set architecture for virtualized workloads. *2021 International Symposium on Secure and Private Execution Environment Design (SEED)* (2021), 121–131.

[19] Sardar, M. U., Musaev, S., and Fetzer, C. Demystifying attestation in intel trust domain extensions via formal verification. *IEEE Access 9* (2021), 83067–83079.

[20] Schuster, F., Costa, M., Fournet, C., Gkantsidis, C., Peinado, M., Mainar-Ruiz, G., and Russinovich, M. Vc3: Trustworthy data analytics in the cloud using sgx. *2015 IEEE Symposium on Security and Privacy* (2015), 38–54.

[21] Schwarz, B., Debray, S. K., Andrews, G. R., and Legendre, M. P. Plto: A link-time optimizer for the intel ia-32 architecture.

[22] Shen, B.-Y., Hsu, W.-C., and Yang, W. A retargetable static binary translator for the arm architecture. *ACM Trans. Archit. Code Optim. 11*, 2 (jun 2014).

[23] Sinha, R., Rajamani, S. K., Seshia, S. A., and Vaswani, K. Moat: Verifying confidentiality of enclave programs. *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security* (2015).

[24] Unnibhavi, H., Cerdeira, D., Barbalace, A., Santos, N., and Bhatotia, P. Secure and policy-compliant query processing on heterogeneous computational storage architectures. *Proceedings of the 2022 International Conference on Management of Data* (2022).

[25] Zimba, A., Chishimba, M., and Chihana, S. A ransomware classification framework based on file-deletion and file-encryption attack structures. *ArXiv abs/2102.10632* (2021).