Compatibility of trusted computing

Qi Hu

01/11/2022

Keywords: Trusted Computing, Binary Rewriting, Distributed System, Software-hardware Co-optimization.

1 Introduction

With the development of cloud computing and big data technologies, more and more applications are being developed for cloud platforms and third-party data centers. However, cloud platform applications often receive various threats, especially the leakage of critical data, which makes many companies reluctant to use the cloud platform. Fortunately, trusted computing prevents confidential violation and protects their applications running on shared servers. In recent years, cloud computing service companies supported confidential computing and provided corresponding Trusted Execution Environment (TEE). For example, Amazon Nitro system [8] uses hardware-based memory isolation to protect data, and Azure [6] provides computing environments for Intel SGX and AMD SEV-SNP and confidential container computing.

How to migrate original applications to a trusted environment becomes another vital issue. Although both SGX and TrustZone provide their SDKs, it requires programmers to divide the trusted and untrusted parts. Glamdring [20] can split the code into untrusted and trusted parts based on tagged data. Occlum [38] brings the LibOSes into SGX to support legacy programs. SCONE [5] provides a secure C standard library interface that allows applications to run in secure containers. These solutions usually require source code and recompile (Glamdring) or the entire program to be run in an enclave (Occlum and SCONE). For those programs where the source code is not available, using Glamdring is not feasible. In addition, putting the entire program into the enclave increases the size of the TCB and expands the attack surface, which is also an unacceptable solution.

Another obstacle is the performance of trusted computing. Switchless Calls [42] changes synchronous execution to asynchronous execution, which can reduce enclave switching. VAULT [41] introduces a variable arity unified tree (VAULT), which compresses the Enclave Page Cache (EPC) and saves overhead. Other studies have focused more on reconfigurable trusted hardware, such as TEEOD [29] and BYOTee [4], which use heterogeneous SoC or FPGA to implement some new features. These hardware-related studies are more concerned with new features than performance optimization. CRONUS [16] gives some suggestions on how to modify the hardware to speed up trusted computing but does not implement them.

Therefore, it is a tricky problem to efficiently migrate legacy programs to TEEs, especially those for which source code is unavailable. This research will first explore the algorithms for finding confidential code and data for legacy programs (Section 4.1). Then we will use these results to guide the rewriting tool to modify legacy programs into software that supports trusted computing (Section 4.2). We intend to have our tools support a variety of legacy software, including distributed programs (Section 4.3), and still maintain high execution efficiency with the hardware-assisted (Section 4.4).

2 Research Objectives

1. Design a dynamic analysis/translation tool to support legacy software on trusted computing systems. This tool can be implemented in 3 parts:
   - Dynamic analysis of legacy software points out which parts of the code are more vulnerable to attack and which data are essential and must 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 enclaves.
   - 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 dynamic analysis/translation tool into a distributed system.
3. Explore performance bottlenecks and then design or 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 and reverse engineering cracking [45]. So, it is helpful 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 [24]. It provides the enclave environment, preventing all other software from accessing the code and data inside an enclave. Also, when data leaves the enclave and is 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 [25]. The secure world has multiple privilege levels, just like a virtual machine (VM), which allows an entire trusted software stack to be implemented.

Due to overly complex operations and unacceptable hardware overhead, Intel started to move towards Trust Domain Extensions (TDX) [32, 33], a new trusted computing architecture introduces a separate trusted hypervisor. The interaction between trusted virtual machines and external untrusted environments should be checked by the security check module Shim.

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

3.2 Binary rewriting and binary translation

Binary rewriting is a technique for modifying 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 [37, 36]. Dynamic binary rewriting performs alterations during execution, which can be used for performance analysis [22] and hot code patching [9]. Minimal-invasive rewriting is based on branch granularity. It will perform additional instruction at the original location by rewriting into branch instructions. This is often used to add a new function to the original program [12]. Full-translation rewriting can convert binaries at any instruction and usually lift the original binary code into intermediate translation representations. Some open-source tools, like QEMU [7] and Valgrind [27], use full-translation for binary rewriting.

3.3 Distributed system in Trusted computing

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

In 2015, the first distributed MapReduce system VC3 [35] was proposed, which keeps the code and data confidential, ensures the correctness and completeness of the results. SGX-PySpark [31] was implemented in 2019, and with the help of TEE, it can protect confidential data.

For other systems, such as database, EnclaveDB [30] uses SGX to protect the database engine and ensure high performance. EncDBDB [13] also uses SGX for data security and is optimized for column-oriented in-memory databases.

In recent years, heterogeneous computing systems, such as Computation Storage Architectures (CSA), have also faced data security issues. IronSafe [43] provides a secure processing system for heterogeneous computing storage architectures using a hardware-assisted trusted execution environment.

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. We need to consider three parts: data analysis, recognition algorithm and recognition accuracy.

The first is **data analysis**. The analysis aims 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 [26] uses a combination of static and dynamic identification methods to improve the accuracy of the analysis. However, for dynamic disassembly, some trapped instructions 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 [39] is a detection tool designed by Berkeley that uses automatic theorem proving and information flow analysis methods to discover the possibility of application 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 tool.

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 [14] and QEMU [7]. *Correctness of the analysis*: We also use open-source test cases 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 [20] to obtain an accurate analysis.

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

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

Code protection. We can use the Minimal-invasive translation method for code segments that need to be protected. Similar to the rev.ng [12] and pin [22], 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 during research.

Data protection. Data protection is more complicated 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 enclaves for protection. PtrSplit focuses on C/C++ pointers and identifies pointers that block the generation of partition boundaries [21]. But for global variables, there is no good solution for now. However, global variables are often not recommended for a highly cohesive and low-coupling system [1, 40], 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 distributed system

Our work will explore two areas related to distributed systems, how to support legacy distributed programs and enable tools to run on distributed systems.

Support legacy distributed programs. These distributed legacy programs tend to have more complex features than to ordinary programs. For example, OpenMP [11] and MPI+OpenMP [18], will use mechanisms such as semaphores, message communication, etc., which cause problems for both the identification and transformation of our tools. We will explore and tackle these challenges during our research process.

Tools run on distributed systems. How to run our tools on a distributed platform is a complex work. DQEMU [44] achieves a distributed dynamic binary translation system. It discusses the implementation issues and performance optimization, including data coherence protocol, locking mechanism, system calls, and remote thread migration. We can take the idea of DQEMU and modify our tools to run on a distributed system.

4.4 Optimize the system by software-hardware co-optimization

Whether running secret code in enclaves or using rewriting tools for transformation, they both introduce a significant performance overhead. Many studies investigated the overhead of trusted computing and the corresponding optimization method, including avoiding enclave switches [42] and reduceing page swaps [28, 41]. But these optimizations may be challenging to implement in our tools because it is hard to change the program's original behavior. In addition, binary rewriting also faces significant performance overhead, and existing software optimizations are limited in dealing with these issues [17].

Some studies have proposed several ideas for hardware-assisted acceleration, such as shared memory [16]. So our future work will summarize the existing optimization and search the performance bottlenecks through performance analysis tools, such as Perf, VTune, etc. After we obtain the bottlenecks of the performance, we can summarize the common characteristics and design/modify some hardware modules to speed up our tools and programs similar to the PIE [34].

5 Expected Outcomes and Significance

To evaluate the effectiveness of our work, as we mentioned in section 4.1, we can use a series of open-source test cases and benchmarks, such as SPEC, UnixBench. We will also consider using SGXGauge [19], a test suite specifically designed to test the performance of trusted systems, to measure the performance of our tool. Besides that, we are able to compare our tools with other cross-platform TEEs. For example, we can use HyperEnclave [15], TrustVisor [23], Occlum [38] and Graphene [10] as baseline systems for performance analysis.

Although these cross-platform TEEs propose various solutions to address the legacy software, most modify libraries and put them into the enclave. This could lead to the expansion of the TCB and cause an attack surface expansion.

Our solution uses binary rewriting to place the analyzed confidential code and data into enclaves, hardening the legacy software and reducing the size of TCB. We anticipate our tools for legacy software can reach the following goals:

1. The analysis tool can efficiently and accurately identify vulnerable code and data. Using analysis tool alone can help developers find software weaknesses and flaws.
2. The binary rewriting tool can rewrite legacy software into enclave-protected programs with the help of analysis results.
3. Our tools are extended to support distributed legacy programs.
4. Based on the performance data, we design or modify the hardware. With hardware-assisted, we expect the performance can exceed other cross-platform TEEs or even approach the performance of native programs.

6 Study Schedule

My study mainly focus on the migration of legacy software in trusted systems, and my arrangement is as follows:

- 1^{st} Year (2023-2024)
  1. Complete the core courses and choose the elective courses that will help me with my research.
  2. Read the paper and design the whole system, while revising our current ideas and plans.
- 2^{nd} Year (2024-2025)
  1. Start to implement our analysis and rewrite tools based on first-year design.
  2. Publish the first paper about our analysis tools.
- 3^{rd} Year (2025-2026)
  1. Modify and optimize our rewriting tools.
  2. Extend our tools to support distributed programs.
  3. Publish the second papers about our rewriting tools.
- 4^{th} Year (2025-2026)
  1. Review our entire system and analyze their performance bottlenecks.
  2. Try to improve the performance of migrated programs by modifying the hardware.
  3. Publish the third papers about our rewriting tools.
  4. Complete the graduation thesis.

References

[1] Global variables are bad. https://kidneybone.com/c2/wiki/GlobalVariablesAreBad, 2013.

[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] Armanuzzaman, M., and Zhao, Z. Byotee: Towards building your own trusted execution environments using fpga. *ArXiv abs/2203.04214* (2022).

[5] Arnautov, S., Trach, B., Gregor, F., Knauth, T., Martin, A., Priebe, C., Lind, J., Muthukumaran, D., O'Keeffe, D., Stillwell, M., Goltzsche, D., Eyers, D., Kapitza, R., Pietzuch, P. R., and Fetzer, C. Scone: Secure linux containers with intel sgx. In *OSDI* (2016).

[6] Azure. Azure confidential computing. https://learn.microsoft.com/en-us/azure/confidential-computing/, 2022.

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

[8] Brown, D. Confidential computing: an aws perspective. https://aws.amazon.com/cn/blogs/security/confidential-computing-an-aws-perspective/, 2021.

[9] 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.

[10] che Tsai, C., Porter, D. E., and Vij, M. Graphene-sgx: A practical library os for unmodified applications on sgx. In *USENIX Annual Technical Conference* (2017).

[11] Dagum, L., and Menon, R. Openmp: an industry standard api for shared-memory programming.

[12] 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).

[13] 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.

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

[15] Jia, Y., Liu, S., Wang, W.-H., Chen, Y., jun Zhai, Z., Yan, S., and He, Z. Hyperenclave: An open and cross-platform trusted execution environment. In *USENIX Annual Technical Conference* (2022).

[16] Jiang, J., Qi, J., Shen, T., Chen, X., Zhao, S., Wang, S., Chen, L., Zhang, G., Luo, X., and Cui, H. Cronus: Fault-isolated, secure and high-performance heterogeneous computing for trusted execution environment. *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)* (2022), 124–143.

[17] Kim, H.-S., and Smith, J. E. Hardware support for control transfers in code caches. *Proceedings. 36th Annual IEEE/ACM International Symposium on Microarchitecture, 2003. MICRO-36.* (2003), 253–264.

[18] Klinkenberg, J., Samfass, P., Bader, M., Terboven, C., and Müller, M. S. Chameleon: Reactive load balancing for hybrid mpi+openmp task-parallel applications. *J. Parallel Distributed Comput. 138* (2020), 55–64.

[19] Kumar, S., Panda, A., and Sarangi, S. R. A comprehensive benchmark suite for intel sgx. *ArXiv abs/2205.06415* (2022).

[20] 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).

[21] 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).

[22] 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).

[23] McCune, J. M., Li, Y., Qu, N., Zhou, Z., Datta, A., Gligor, V. D., and Perrig, A. Trustvisor: Efficient tcb reduction and attestation. *2010 IEEE Symposium on Security and Privacy* (2010), 143–158.

[24] 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).

[25] 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.

[26] 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.

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

[28] Orenbach, M., Lifshits, P., Minkin, M., and Silberstein, M. Eleos: Exitless os services for sgx enclaves. *Proceedings of the Twelfth European Conference on Computer Systems* (2017).

[29] Pereira, S. A. G., Cerdeira, D., Rodrigues, C., and Pinto, S. Towards a trusted execution environment via reconfigurable fpga. *ArXiv abs/2107.03781* (2021).

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

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

[32] 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.

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

[34] Schneider, M., Dhar, A., Puddu, I., Kostiainen, K., and Capkun, S. Pie: A platform-wide tee.

[35] 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.

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

[37] 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).

[38] Shen, Y., Tian, H., Chen, Y., Chen, K., Wang, R., Xu, Y., and Xia, Y. Occlum: Secure and efficient multitasking inside a single enclave of intel sgx. *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems* (2020).

[39] 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).

[40] Smith, F. Global variables are evil and unsafe. https://www.forrestthewoods.com/blog/global-variables-are-evil-and-unsafe/, 2022.

[41] Taassori, M., Shafiee, A., and Balasubramonian, R. Vault: Reducing paging overheads in sgx with efficient integrity verification structures. *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems* (2018).

[42] Tian, H., Zhang, Q., Yan, S., Rudnitsky, A., Shacham, L., Yariv, R., and Milshten, N. Switchless calls made practical in intel sgx. *Proceedings of the 3rd Workshop on System Software for Trusted Execution* (2018).

[43] 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).

[44] Zhao, Z., Jiang, Z., Liu, X., Gong, X., Wang, W., and Yew, P. Dqemu: A scalable emulator with retargetable dbt on distributed platforms. *49th International Conference on Parallel Processing - ICPP* (2020).

[45] 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).