Summary of Changes

We would like to express our gratefulness to the reviewers for your precious time on reviewing our submission. Here, we present the motivation of submitting a journal paper based on our USENIX Security 2017 conference paper and summarize the major updates and improvements. Then, we provide a list of the detailed changes in our IEEE TIFS submission.

I. Changes Between IEEE TIFS Submission and USENIX Security 2017 Paper

In our USENIX Security 2017 conference paper [1], we proposed NINJA, which is a transparent malware analysis system on ARM platform. It reduces the footprints and artifacts of an analysis system and provides a more transparent execution environment for the malware analyst.

However, in the scenario of continuous malware analysis, it is also important to achieve a fast system restoration. Thus, we design a fast restoration mechanism which improves the existing restoration systems [2], [3], [4], [5]. Specifically, we implement selective memory restoration, runtime file system restoration, and complete context restoration. The selective memory restoration leverages hardware-based tracing to record the changed memory addresses and helps to improve the efficiency of memory restoration. The runtime file system restoration uses existing runtime file system switching technology to efficiently switch between alternative file systems, while the complete context restoration performs a complete system register restoration after each analysis session.

Moreover, we extend the trace subsystem of NINJA to support data address tracing. The data address tracing records the related data address of each load/restore instruction and helps an analyst to learn additional information about the current running status of a target malware. As an example, the analyst may easily infer the plaintext or encryption keys in AES algorithm with the data addresses [6]. Moreover, the data address information is also helpful for dynamic taint analysis [7] and the selective memory restoration discussed above. In light of this, we add data address tracing to the trace subsystem and implement it in an NXP i.MX53 Quick Start Board. This implementation leverages Linux as the rich OS instead of Android (which we implemented in our USENIX Security conference version) and shows the OS-agnostic feature of NINJA.

We also improve the instruction tracing with address range and process ID filters. These filters help the analyst focus on the interested instructions and processes. Additionally, we implement more debugging functions related to the CPU speculative execution. The PMU-based functionality may provide more granularities for stepping. Moreover, some of these granularities are related to the speculative execution feature of the CPU which is abused by recent Meltdown [8] and Spectre [9] attacks. With these additional granularities, the analyst can calculate the Meltdown- and Spectre- related events like cache miss ratio, number of speculatively executed instructions, and branch mispredict ratio. These statistic data are proposed to infer Meltdown and Spectre attacks [10], [11].

Some contents are also removed due to conciseness and the page limit.

II. Detailed Changes

A. Abstract

1) Added a discussion of the newly implemented system restoration mechanism.

B. Introduction

1) Added a brief introduction of the improved system restoration.

2) Added the performance evaluation result of our system restoration mechanism.

3) Added a detailed description of the new contributions of this journal extension as compared with our conference paper.

4) Removed the evaluation result of instruction skid.

C. Background

1) Removed the background about ARM architecture.

2) Removed Figure 1: The ARMv8 and ARMv7 architectures.

3) Removed the survey of Futuremark about the ARM processors in popular smartphones and tablets.

D. Related Work

1) Added Section 3.3 to discuss the related works on the system restoration.

3) Revised the related works on transparent malware analysis on x86 in Section 3.1.

2) Removed Section 3.3: TrustZone-related systems.

E. System Architecture

1) Added data address tracing in Section 4.2 (i.e., using another feature of the ETM to achieve the data address tracing).

2) Added CPU speculative execution related example (i.e., the `INST_SPEC` event that fires after an instruction is speculatively executed) in Section 4.3.

F. Design and Implementation

1) Added a new implementation testbed, an NXP i.MX53 Quick Start Board.

2) Added instruction address filter in Section 5.3.1 (i.e., using the address range comparators in the ETM to restrict the trace to specific memory ranges).

3) Added process ID filter in Section 5.3.1 (i.e., using the context ID filters in the ETM to restrict the trace to a specific process).

4) Added Section 5.3.4 to describe the detailed data address tracing.

5) Added 6 more granularities for stepping debug including branch instruction, mispredicted branch instruction, L1 data cache read operation, L1 data cache refill operation, speculatively executing a load instruction, and speculatively executing a branch instruction in Section 5.4.1.

6) Added Table 1: Representative stepping modes in NINJA.

7) Added Section 5.5 to describe the improved system restoration including selective memory restoration, runtime file system restoration, and complete context restoration.

8) Revised Section 5.6 for conciseness.

G. Evaluation

1) Added Section 7.3.2 to evaluate the performance of the newly implemented system restoration.

2) Added Table 5: Time consumption of system restoration.

3) Revised the description in Section 7.2.1 and Section 7.2.2 for conciseness.

4) Removed Section 7.1: Output of tracing subsystem and Section 7.5: Skid evaluation.

5) Removed Figure 7: Accessing system instruction interface and Figure 8: Memory mapped interface.

H. Conclusion

1) Added a description of the improved system restoration mechanism.

I. Appendix

1) Removed the appendix section.

References

[1] Z. Ning and F. Zhang, "Ninja: Towards transparent tracing and debugging on ARM," in *Proceedings of 26th USENIX Security Symposium (USENIX Security'17)*, 2017.

[2] D. Kirat, G. Vigna, and C. Kruegel, "Barebox: Efficient malware analysis on bare-metal," in *Proceedings of the 27th Annual Computer Security Applications Conference (ACSAC'11)*, 2011.

[3] Kirat, Dhilung and Vigna, Giovanni and Kruegel, Christopher, "Barecloud: Bare-metal analysis-based evasive malware detection," in *Proceedings of the 23rd USENIX Security Symposium (USENIX Security'14)*, 2014.

[4] F. Zhang, K. Leach, A. Stavrou, and H. Wang, "Using hardware features for increased debugging transparency," in *Proceedings of The 36th IEEE Symposium on Security and Privacy (S&P'15)*, 2015, pp. 55–69.

[5] L. Guan, S. Jia, B. Chen, F. Zhang, B. Luo, J. Lin, P. Liu, X. Xing, and L. Xia, "Supporting transparent snapshot for bare-metal malware analysis on mobile devices," in *Proceedings of the 33rd Annual Computer Security Applications Conference (ACSAC'17)*, 2017.

[6] M. Lipp, D. Gruss, R. Spreitzer, C. Maurice, and S. Mangard, "ARMageddon: Cache attacks on mobile devices," in *Proceedings of the 25th USENIX Security Symposium (USENIX Security'16)*, 2016.

[7] E. J. Schwartz, T. Avgerinos, and D. Brumley, "All you ever wanted to know about dynamic taint analysis and forward symbolic execution (but might have been afraid to ask)," in *Proceedings of 31st IEEE Symposium on Security and Privacy (S&P'10)*, 2010.

[8] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg, "Meltdown," *arXiv preprint arXiv:1801.01207*, 2018.

[9] P. Kocher, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom, "Spectre attacks: Exploiting speculative execution," *arXiv preprint arXiv:1801.01203*, 2018.

[10] E. Israel, D. Marx, Y. Alon, A. Gafni, and B. Omelchenko, "Detection of the Meltdown and Spectre vulnerabilities," https://research.checkpoint.com/detection-meltdown-spectre-vulnerabilities-using-checkpoint-cpu-level-technology/.

[11] Capsule8, "Part two: Detecting Meltdown and Spectre by detecting cache side channels," https://capsule8.com/blog/detecting-meltdown-spectre-detecting-cache-side-channels/.