Code Attestation for Monitor Compromise Detection

Matthew Mifsud and Christian Colombo[0000−0002−2844−5728]

Department of Computer Science, University of Malta, Malta
{matthew.mifsud.22,christian.colombo}@um.edu.mt

Abstract. Runtime monitors detect deviations from expected behaviour, making them effective in detecting malicious activity. Their effectiveness, however, assumes the monitor itself remains uncompromised. This work addresses the threat of in-memory code tampering by proposing a detection mechanism based on remote code attestation.

Keywords: Runtime monitoring · Cryptography · Code attestation

1 Introduction

Protecting systems from malicious behaviour requires detecting unexpected behaviour during execution. This can be achieved using online runtime monitors. However, because monitors run alongside the systems they observe, they become attractive targets for adversaries. A common method of compromise involves modifying code in memory, allowing an attacker to tamper with the monitor and suppress detection. Maintaining the monitor's integrity during execution therefore requires a mechanism capable of continuously verifying its code state.

The threat of monitor compromise is becoming increasingly significant, as runtime monitors are now deployed across a wide range of environments. These include cloud platforms [1], mobile operating systems [2], industrial control systems [3], financial transaction systems [4], and other environments where trust in the monitor's behaviour is essential. Compromising a monitor can allow malicious behaviour to go undetected, leading to data breaches, service disruption, or unsafe operation in critical systems. Despite their importance, securing monitors themselves seems not to be so well studied [5].

Among the various threats to monitor integrity, a particularly concerning class is in-memory code tampering. Documented techniques show that attackers can disable or bypass programs by directly modifying their code during execution. These attacks include code injection [6–8], where new code is introduced at runtime, and code manipulation [9–11], where existing instructions are altered. Such attacks are especially effective in environments that allow writable memory or support runtime code inspection and modification. These include, for example, native applications where low-level techniques such as inline hooking [12] can overwrite function entry points to redirect execution, and managed runtimes like the Java Virtual Machine, which allow programs to inspect their own classes and dynamically load new code during execution.

Verifying the state of a runtime monitor's code during execution is particularly challenging in adversarial environments. An attacker with sufficient privileges may not only tamper with the monitoring logic to suppress detection but also interfere with any local mechanism responsible for ensuring its integrity. In such cases, the system can no longer be trusted to perform verification reliably. To provide meaningful assurance, verification must be performed by an external entity that remains beyond the attacker's control and can independently assess the monitor's code state. This requires generating runtime proofs that accurately reflect the executing code. However, this process is inherently difficult, as code in memory is subject to dynamic behaviours such as compiler optimisations, platform-specific representations, and deliberate changes introduced by trusted instrumentation tools. These factors introduce variability in how code appears at runtime, complicating the task of distinguishing legitimate modifications from malicious ones.

2 Proposed Solution

To address the challenge of detecting when a runtime monitor has been tampered with during execution, this work proposes a remote code attestation mechanism. Remote attestation [13] is a cryptographic technique in which proofs of a program's code state are periodically generated and sent to an external verifier. In our approach, the verifier issues unpredictable challenges, and the monitor responds with a proof based on a measurement of its in-memory code.

While the proposed mechanism is applicable to various runtime environments, this work targets the Java Virtual Machine (JVM) for both design and implementation. The JVM is widely adopted and supports dynamic code modification, making it a representative platform for demonstrating the technique. Instead of measuring native machine code, whose format and location can vary across platforms, this work focuses on Java bytecode, which remains structured, accessible, and consistently resident in memory throughout execution. Bytecode also serves as the authoritative representation from which all execution in the JVM is derived, including just-in-time compiled machine code. These characteristics make it particularly susceptible to tampering through standard JVM capabilities such as dynamic class loading and bytecode instrumentation.

This work presents a practical design and implementation of a remote code attestation mechanism for detecting in-memory tampering of runtime monitors within the JVM. The mechanism detects a range of attacks, from subtle code modifications to the complete disabling of monitoring logic or the attestation process itself, as well as network-level threats such as proof forgery. We also evaluated an optimisation inspired by the pseudorandom memory traversal technique used in the SWATT attestation scheme [14]. In our approach, we reduce overhead by attesting a pseudorandom subset of in-memory code, introducing a tunable trade-off between performance and security. Overall, this work lays a foundation for securing runtime monitors against in-memory tampering and presents an approach that can be adapted to other platforms.

References

1. Cotroneo, D., De Simone, L., Liguori, P., Natella, R.: Run-time failure detection via non-intrusive event analysis in a large-scale cloud computing platform. *J. Syst. Softw.* **198**, 111611 (2023). https://doi.org/10.1016/j.jss.2023.111611
2. Chircop, L., Colombo, C., Pace, G.J.: Device-centric monitoring for mobile device management. *Electron. Proc. Theor. Comput. Sci.* **205**, 31–44 (2016). https://doi.org/10.4204/EPTCS.205.3
3. Janicke, H., Nicholson, A., Webber, S., Cau, A.: Runtime-monitoring for industrial control systems. *Electronics* **4**(4), 995–1017 (2015). https://doi.org/10.3390/electronics4040995
4. Colombo, C., Pace, G.J.: Considering academia-industry projects meta-characteristics in runtime verification design. In: Margaria, T., Steffen, B. (eds.) *Leveraging Applications of Formal Methods, Verification and Validation. Industrial Practice – ISoLA 2018*. Lecture Notes in Computer Science, vol. 11247, pp. 32–41. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-03427-6_5
5. Colombo, C., Curmi, A., Abela, R.: RVsec: Towards a comprehensive technology stack for secure deployment of software monitors. In: *Proc. of the 7th ACM Int. Workshop on Verification and Monitoring at Runtime Execution (VORTEX)*, pp. 13–18. ACM, Vienna, Austria (2024). https://doi.org/10.1145/3679008.3685542
6. Holzinger, P., Triller, S., Bartel, A., Bodden, E.: An in-depth study of more than ten years of Java exploitation. In: *Proc. of the 2016 ACM SIGSAC Conf. on Computer and Communications Security (CCS)*, pp. 779–790. ACM, Vienna, Austria (2016). https://doi.org/10.1145/2976749.2978361
7. Poeplau, S., Fratantonio, Y., Bianchi, A., Kruegel, C., Vigna, G.: Execute this! Analyzing unsafe and malicious dynamic code loading in Android applications. In: *Proc. of the Network and Distributed System Security Symposium (NDSS)*. Internet Society, San Diego, CA, USA (2014).
8. Sharma, A., Wittlinger, M., Baudry, B., Monperrus, M.: SBOM.EXE: Countering dynamic code injection based on software bill of materials in Java. *arXiv preprint* arXiv:2407.00246 (2024). https://doi.org/10.48550/arXiv.2407.00246
9. Barbu, G., Thiebeauld, H., Guerin, V.: Attacks on Java Card 3.0 combining fault and logical attacks. In: Guilley, S. (ed.) *Smart Card Research and Advanced Applications – CARDIS 2010*. Lecture Notes in Computer Science, vol. 6035, pp. 148–163. Springer, Passau, Germany (2010). https://doi.org/10.1007/978-3-642-12510-2_11
10. Park, T., Lettner, J., Na, Y., Volckaert, S., Franz, M.: Bytecode corruption attacks are real—and how to defend against them. In: Bilge, L., Balzarotti, D. (eds.) *Detection of Intrusions and Malware, and Vulnerability Assessment – DIMVA 2018*. Lecture Notes in Computer Science, vol. 10885, pp. 326–348. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-93411-2_15
11. Dabirsiaghi, A.: JavaSnoop: How to Hack Anything in Java. Whitepaper, *Black Hat USA*, Las Vegas, USA (2010). https://media.blackhat.com/bh-us-10/whitepapers/Dabirsiaghi/BlackHat-USA-2010-Dabirsiaghi-JavaSnoop-wp.pdf
12. Yavo, O., Deshotels, L., Lindner, J.: Captain Hook: Pirating AVs to bypass exploit mitigations. *Black Hat USA* (2016). https://www.blackhat.com/docs/us-16/materials/us-16-Yavo-Captain-Hook-Pirating-AVs-To-Bypass-Exploit-Mitigations-wp.pdf
13. Banks, A.S., Kisiel, M., Korsholm, P.: Remote attestation: A literature review. *arXiv preprint* arXiv:2105.02466 (2021). https://doi.org/10.48550/arXiv.2105.02466

14. Seshadri, A., Perrig, A., van Doorn, L., Khosla, P.: SWATT: Software-based attestation for embedded devices. In: *Proc. of the IEEE Symp. on Security and Privacy*, pp. 272–282. IEEE, Oakland, CA, USA (2004). https://doi.org/10.1109/SECPRI.2004.1301329