Hardware-Enforced Integrity and Provenance for Distributed Code Deployments

Area: Guidelines for Software Integrity Chains and Provenance

MARCELA S. MELARA, Research Scientist, Intel Labs, USA
MIC BOWMAN, Senior Principal Engineer, Intel Labs, USA

1 Introduction

Distributed code deployments today rely very heavily on a complex series of transformation and inspection operations, called the *software supply chain*, for the creation of an *executable bundle* that is run at a cloud provider. For example, a compilation *tool* transforms one or more input *software artifacts* (i.e., source code files and shared libraries) generating one or more output artifacts (i.e., bytecode or binary files, other shared libraries, container images). These may then subsequently be passed as input artifacts to an automated testing tool, inspecting the functionality of the artifacts without modifying them, and returning the test results. Thus, we refer to a software artifact and attached metadata as an executable bundle that can be deployed at a cloud provider.

However, a plethora of attacks have undermined the integrity of entire deployments by compromising one or more operations of the software supply chain [2–4, 6, 7, 10, 13, 15, 17–19]. Prior work [5, 8, 9, 21] has proposed addressing these issues by capturing metadata about software artifacts to obtain verifiable information about the supply chain of an application. in-toto [21], in particular, realizes this approach relying on cryptographically verifiable metadata about each individual supply chain operation to ensure that an executable bundle was indeed created by the expected supply chain tools in the expected order.

Yet, as distributed computing continues to evolve, we observe three major trends that introduce new security challenges to code deployment.

1. Rise of the microservices. Rather than deploying a monolithic, self-contained, fully packaged application, developers increasingly decompose an application into small, independent components. These microservices are then uploaded at a cloud provider enabling greater application portability, efficiency and fault isolation.

2. Shift towards lightweight containers. As cloud providers seek to optimize resource usage while improving microservice performance they employ lighter weight execution environments that rely on software-based techniques for isolation in a multi-tenant setting (e.g., [16, 20]). However, this means cloud providers can no longer rely on virtual machine-like mechanisms to protect their resources against vulnerable or buggy code, and strongly isolate co-tenant microservices.

3. On-demand deployment. To further optimize resource usage, hosting services determine on-demand which compute platforms are most suitable to deploy particular microservices.

The core problem. These three trends demonstrate that today's distributed applications require more adaptable and comprehensive means to establish trust in deployed code. That is, it is no longer sufficient to only ensure that code is generated by the expected software supply chain.

Deployed microservices must adhere to a multitude of application-level security requirements and regulatory constraints imposed by mutually distrusting *application principals*– software developers, cloud providers, and even data owners. Although these principals wish to enforce their individual security requirements, they do not currently have a common way of easily identifying, expressing and automatically enforcing these requirements at deployment time.

Our Proposal. CDI (Code Deployment Integrity) is a security policy framework that enables distributed application principals to establish trust in deployed code through *high-integrity* provenance information.

We observe that principals expect the software supply chain to preserve certain code security properties throughout the creation of an executable bundle, even if the code is transformed or inspected through various tools (e.g., compilation inserts stack canaries for memory safety). Our key insight in designing CDI is that even if application principals do not trust each other directly, they can trust a microservice bundle to meet their security policies *if they can trust the tools involved in creating the bundle*.

Authors' addresses: Marcela S. Melara, Intel Labs, Hillsboro, OR, USA, marcela.melara@intel.com; Mic Bowman, Intel Labs, Hillsboro, OR, USA, mic.bowman@intel.com.

2 Code Deployment Integrity

As a microservice is created through a software supply chain, each tool in the supply chain digitally signs its input and output as well as operation-specific metadata (e.g., compiler flags, testing configuration). Crucially, CDI metadata also embeds a cryptographic hash of the metadata attached to each input artifact generated by "upstream" operations forming a hash tree that represents an auditable *provenance chain* for the entire supply chain of the executable microservice bundle received by the cloud provider.

2.1 Key Components

Protecting the integrity of the CDI metadata and efficiently auditing the provenance chain is vital to the security and adoptability of CDI. We address these important challenges with two key mechanisms.

1. Trusted execution environments (TEEs). CDI leverages TEEs (e.g., [1, 11, 12, 14]) for their *hardware-enforced* integrity and authentication properties for code and data. Specifically, running individual software supply chain tools in a TEE such as Intel SGX [11] can help CDI extend and enforce provenance in hardware. As tools derive their signing keys within an enclave, the digitally signed CDI metadata is bound to the specific platform that generated the keys.

2. Vetting authorities. CDI introduces independent entities called vetting authorities that are responsible for certifying that specific software supply chain tools preserve the expected security properties. Specifically, CDI requires vetting authorities to produce digitally signed tool certifications that can be embedded in a tool's provenance metadata. Additionally, vetting authorities could leverage TEE technology to remotely authenticate the TEE-enabled tools they are certifying.

In practice, we envision each vetting authority will determine the exact process by which to vet individual supply chain tools. To enable vetting a large array of supply chain tools available to developers and cloud providers today, CDI supports a hierarchy of vetting authorities akin to how traditional certificate authorities operate today.

2.2 Design Goals

Our design of CDI meets the following three goals.

G1: Code provenance can be validated without revealing any software artifacts. Only the cryptographic hashes (e.g., SHA-256) of both input and output artifacts are included in CDI metadata. Thus, application principals need not share proprietary or highly privacy-sensitive artifacts directly with other principals to enable validation.

G2: Application principals need not rely on a single centralized root of trust. Developers, cloud providers and data owners may independently select the vetting authorities they trust to properly certify tools in the software supply chain ecosystem. Further, CDI allows principals to require a threshold number of trusted vetting authorities to have certified a given tool for added confidence in its operation. At validation time, CDI ensures that a microservice's provenance meets all specified trust policies.

G3: Trust in code can be established at deployment time without a priori knowledge of the tools that created the code. As long as a chain of trust between a vetting authority and a given tool can be established, the CDI provenance metadata is sufficient to gain trust in a microservice and ensure it enforces a principal's security policy.

3 Conclusion

We have presented CDI (Code Deployment Integrity), a framework for verifiably capturing and validating microservice security properties and requirements via high-integrity metadata about the software supply chain. Our goal is to enable security-oriented code deployment, in which provenance metadata can be used to enforce complex security policies imposed by a multitude of application principals. By leveraging trusted execution environments and vetting authorities that act as trusted intermediaries, CDI provides strong integrity for code provenance metadata while reducing adoption efforts for software developers compared to prior approaches.

References

[1] Advanced Micro Devices, Inc. [n.d.]. AMD Secure Encrypted Virtualization (SEV). https://developer.amd.com/sev/.

[2] Claudio A. Ardagna, Rasool Asal, Ernesto Damiani, and Quang Hieu Vu. 2015. From Security to Assurance in the Cloud: A Survey. *ACM Comput. Surv.* 48, 1, Article 2 (July 2015), 50 pages. https://doi.org/10.1145/2767005

[3] V. Craciun, P. A. Felber, A. Mogage, E. Onica, and R. Pires. 2020. Malware in the SGX supply chain: Be careful when signing enclaves! *IEEE Transactions on Dependable and Secure Computing* (2020), 1–1. https://doi.org/10.1109/TDSC.2020.3024562

[4] Ruian Duan, Omar Alrawi, Ranjita Pai Kasturi, Ryan Elder, Brendan Saltaformaggio, and Wenke Lee. 2020. Towards Measuring Supply Chain Attacks on Package Managers for Interpreted Languages. arXiv:2002.01139 [cs.CR]

[5] Stephen Elliot. 2017. Introducing Grafeas: An open-source API to audit and govern your software supply chain. Google Cloud Blog. https://cloud.google.com/blog/products/gcp/introducing-grafeas-open-source-api-

[6] FireEye. 2020. Highly Evasive Attacker Leverages SolarWinds Supply Chain to Compromise Multiple Global Victims With SUNBURST Backdoor. FireEye Blogs - Threat Research. https://www.fireeye.com/blog/threat-research/2020/12/evasive-attacker-leverages-solarwinds-supply-chain-compromises-with-sunburst-backdoor.html

[7] Sean Gallagher. 2016. Rage-quit: Coder unpublished 17 lines of JavaScript and "broke the Internet". Ars Technica. https://arstechnica.com/information-technology/2016/03/rage-quit-coder-unpublished-17-lines-of-javascript-and-broke-the-internet/

[8] Google. [n.d.]. Binary Authorization. https://cloud.google.com/binary-authorization/, retrieved Sep 2020.

[9] Google. 2018. Kritis: Deployment Authorization for Kubernetes Applications. https://github.com/grafeas/kritis/blob/master/docs/binary-authorization.md.

[10] Volker Gruhn, Christoph Hannebauer, and Christian John. 2013. Security of Public Continuous Integration Services. In *Proceedings of the 9th International Symposium on Open Collaboration* (Hong Kong, China) *(WikiSym '13)*. Association for Computing Machinery, New York, NY, USA, Article 15, 10 pages. https://doi.org/10.1145/2491055.2491070

[11] Intel Corporation. [n.d.]. Intel Software Guard Extensions (Intel SGX). https://software.intel.com/en-us/sgx.

[12] Intel Corporation. [n.d.]. Intel Trust Domain Extensions (Intel TDX). https://software.intel.com/content/www/us/en/develop/articles/intel-trust-domain-extensions.html.

[13] E. Levy. 2003. Poisoning the software supply chain. *IEEE Security Privacy* 1, 3 (2003), 70–73.

[14] Frank McKeen, Ilya Alexandrovich, Alex Berenzon, Carlos V. Rozas, Hisham Shafi, Vedvyas Shanbhogue, and Uday R. Savagaonkar. 2013. Innovative Instructions and Software Model for Isolated Execution. In *Proc. Hardware and Architectural Support for Security and Privacy*.

[15] Marc Ohm, Henrik Plate, Arnold Sykosch, and Michael Meier. 2020. Backstabber's Knife Collection: A Review of Open Source Software Supply Chain Attacks. In *Detection of Intrusions and Malware, and Vulnerability Assessment*, Clémentine Maurice, Leyla Bilge, Gianluca Stringhini, and Nuno Neves (Eds.). Springer International Publishing, Cham, 23–43.

[16] Pat Hickey. 2019. Announcing Lucet: Fastlys native WebAssembly compiler and runtime. https://www.fastly.com/blog/announcing-lucet-fastly-native-webassembly-compiler-runtime. Accessed 10 Feb 2021.

[17] C. Paule, T. F. Dllmann, and A. Van Hoorn. 2019. Vulnerabilities in Continuous Delivery Pipelines? A Case Study. In *2019 IEEE International Conference on Software Architecture Companion (ICSA-C)*. 102–108.

[18] Thomas Ristenpart, Eran Tromer, Hovav Shacham, and Stefan Savage. 2009. Hey, You, Get off of My Cloud: Exploring Information Leakage in Third-Party Compute Clouds. In *Proceedings of the 16th ACM Conference on Computer and Communications Security* (Chicago, Illinois, USA) *(CCS '09)*. Association for Computing Machinery, New York, NY, USA, 199212. https://doi.org/10.1145/1653662.1653687

[19] Isaac Z. Schlueter. 2016. kik, left-pad, and npm. The npm Blog. https://blog.npmjs.org/post/141577284765/kik-left-pad-and-npm

[20] Simon Shillaker and Peter Pietzuch. 2020. Faasm: Lightweight Isolation for Efficient Stateful Serverless Computing. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*. USENIX Association, 419–433. https://www.usenix.org/conference/atc20/presentation/shillaker

[21] Santiago Torres-Arias, Hammad Afzali, Trishank Karthik Kuppusamy, Reza Curtmola, and Justin Cappos. 2019. in-toto: Providing farm-to-table guarantees for bits and bytes. In *USENIX Security '19*.