Attestable Software Versioning for Stateful Confidential Computing

Charly Castes
EPFL, Switzerland

Edouard Bugnion
EPFL, Switzerland

ABSTRACT

Trusted execution environments enable the creation of confidential and attestable enclaves that exclude the platform and service providers from the trusted base. From its initial attestable state, a stateful enclave such as a confidential database can hold confidential information in memory or use an enclave-specific secret seed to encrypt it on disk. The attestation logic is bound to a unique software version, and does not provide a mechanism to upgrade software version.

We propose *attestable software versioning* to ensure the trustworthy software migration of stateful enclaves in the context of an untrusted service operator. Attestable software versioning relies on *extended attestation*, a two-steps hashing process for measurement validation of an enclave extended with its complete software lineage, which further restricts migration to white-listed software versions. Enclaves rely on mutual local or remote extended attestation during the software upgrade; client program use remote extended attestation to determine the software lineage decisions made by the untrusted service operator. The mechanism enables a full separation of roles and responsibilities between software editors ,which cannot access data, and untrusted platform operators, who trigger attestable software upgrades.

ACM Reference Format:
Charly Castes and Edouard Bugnion. 2022. Attestable Software Versioning for Stateful Confidential Computing. In *Proceedings of Proceedings of the 5th Workshop on System Software for Trusted Execution (SysTEX '22 Workshop).* ACM, New York, NY, USA, 3 pages.

1 INTRODUCTION

Trusted Execution Environments (TEE) such as Intel SGX [8], AMD SEV-SNP [16], AMD TrustZone [13] or Keystone [10] enable the creation of isolated software environments that are protected from the outside world, including all system software such as a hypervisors, operating systems, and utilities available to system administrators.

The integrity of initial code and data is guaranteed inside a TEE, as well as the confidentiality of any subsequently produced or injected state. An *attestation mechanism* [1] enables clients to identify a TEE by its initial state and verify that it runs on top of a trusted hardware. After attestation, if the TEE is deemed trustworthy, the client can send sensitive data over a secure channel [9] without the need to trust the platform or service provider operators.

A TEE is identified by its *measurement* [1], a hash over its initial code and data, that can be linked to human-readable source code using a reproducible build system. The measurement alone is enough information for trusting stateless applications or applications that manage only soft state in memory. The mechanism is however insufficient to handle software upgrades as the state of the service must, in the general case, be securely transferred from one version of the software to another. This has the unfortunate consequence of preventing the transparent software upgrade of stateful services such as confidential databases [3, 5, 12, 14] when the service operator cannot be trusted. Indeed, the traditional software upgrade approach based on state externalization could be trivially abused by a malicious operator to either leak data or inject corrupt data into the restarted enclave.

Some stateful confidential services work around the problem by having the client program drive data migration whenever the confidential service software must be upgraded. As an example, the Signal messaging application's secure value recovery service purposefully requires *all* the clients to migrate their own keys to a new enclave to complete a software upgrade [12].

We propose *attestable software versioning* to ensure the trustworthy software migration of stateful enclaves. By separating the roles and responsibilities of the software editor, which publishes reproduceable builds and never has access to data, and of the untrusted service operator responsible for the ongoing operation of the service including the initiation of software upgrades, we provide end users transparency guarantees on the lineage of software through remote attestation.

2 DESIGN

Enabling software update for stateful confidential services operated by an untrusted service operator poses issues regarding:

(1) **The provenance of internal state**. The service operator must be constrained to import only authorized state into the service. In addition, the provenance of data should be attestable so that the end user can decide whether or not it deems the service trustworthy.
(2) **The migration of internal state**. The service operator must be constrained to migrate the service's state only to authorized versions. In particular, the end user must be aware of all data migration and, depending on the security requirements and trust level of the operator, be able to prevent migration to a particular version.

We propose to solve both issues by extending attestation with the history of software versions and controlling data migration using a trusted white list of authorized versions.

Figure 1 describes the workflow for attestable software versioning. A *software editor* publishes verifiable source code and the corresponding artifacts obtained through a reproducible build system.

The untrusted *operator* plays an active role in the data migration, which consists of the following steps: (1) the operator selects a subset of versions and commits their measurements to a white list operated by an external trusted party. (2) the operator then launches a new enclave ("V_5" in Figure 1) with as initial state the new enclave extended with the current version history. (3) The new enclave

SysTEX '22 Workshop, @ASPLOS, Lausanne, Switzerland
2022.

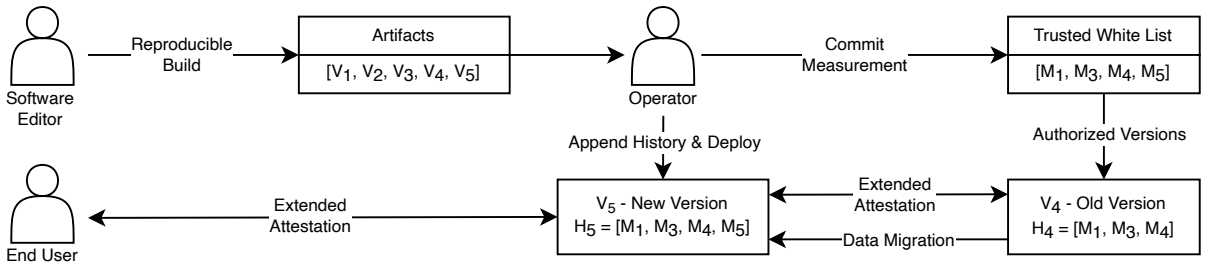

Figure 1: Attestable Software Versioning

establishes a secure, attested channel with the old enclave. Through extended attestation, each enclave can validate the integrity of software published on the white list and their respective lineages. The new lineage must extend the current lineage. (4) Migration begins at the successful completion of the previous step. (5) The new enclave takes over the service.

The *end user* queries the lineage of an enclave through extended attestation whenever it establishes a connection with the service.

2.1 Threat Model

We follow the threat model of other work in confidential computing [2, 4, 7, 11] in which an adversary with full administrative privileges tries to access confidential data or to damage the enclave's integrity. Specifically, the operator may be the attacker.

The *software editor* publishes source code and corresponding artifacts obtained through a reproducible build system. We trust that the published software does not contain directly-exploitable bugs or backdoor. Any breach of trust, including collusion with the service operator, would be evident in the published artifacts.

Through remote attestation, end users receive a proof of the full history of software migrations, and can use that information before sharing any new data with the service.

Denial-of-service attacks, a known limitation of most TEEs including SGX [8], and hardware side channels [6] are out of scope. We assume a correct underlying implementation of the hardware that provides confidentiality, integrity, and attestation.

2.2 Extended Attestation Mechanism

The attestation is extended by incorporating the history in the measurement, this is done by appending history pages to the initial state of the application. The measurement therefore corresponds to the hash over the enclave's code and data plus the history pages. During attestation the enclave sends its history to the client.

The history is validated using a two steps hashing process. First, the service operator initializes a hash at build time and updates it with the code and data of the application (excluding the history), but does not finalize it yet: instead, it saves the hash's internal state and includes it within the history pages. Second, at attestation time, the client initializes a new hash with the values previously computed. At that point the client can: (1) finalize the hash to get the measurement of the enclave if there were no history added, we call it the *code measurement*, and (2) extend the hash with the history and finalize it to get the actual measurement (including history pages). If the computed measurement matches the measurement from the standard attestation process, the client has successfully learned the code measurement and verified the authenticity of the history (it's the same as the one burnt in the enclave pages). Note that the two steps hashing process is secure assuming second-preimage resistance [15] of the underlying hash function.

The extended attestation is a mechanism to attest that a given blob of data is written on the last few memory pages of an enclave, but do not enforce any semantic for that data. For the history to be meaningful the enclaves must take the appropriate actions.

2.3 Data Migration Policy

We define the history of an enclave as an ordered list of *code measurements*: $H = [M_1, ..., M_N]$. We say that H_A is included in H_B if each measurement in H_A is also present in H_B, and the relative order of measurements in H_A is preserved in H_B.

To solve the problem of attestable provenance of the system's state, a version A accepts to migrate its data to a version B if and only if the history of enclave A, H_A, is included in the history of enclave B, H_B. This policy enforces that the internal state of an application can only be derived from the internal states of previous versions, according to the application's history.

This policy alone does not prevent the service operator from migrating the data to a flawed version with the appropriate history. Therefore, we introduce an ordered white list of authorized versions W_L. The refined policy is the following: enclave A accepts to migrate its data to an enclave B if and only if $H_\mathsf{A} \subset H_\mathsf{B}$ and $H_\mathsf{B} \subset W_L$.

The white list must be trusted by the end user. It should be organized as a tamper-proof chain of immutable content, *e.g.,* a log operated by a trusted third party or built on top of a distributed ledger. The service operator must commit a version's code measurement to the chain before being able to migrate data. The end user can either decide to manually approve versions or rely on the service operator's accountability through its public commitments.

3 CURRENT WORK

We are building a prototype that implements all aspects of attestable software versioning as described in §2, including a toolchain to launch enclaves with an extended initial state, the library that runs within each enclaves to implement mutual extended attestation, a client library, and an optional data migration mechanisms that encrypts secondary storage across attestable software upgrades.

Our prototype targets existing SGX-enabled hardware, even though the design can be applied to other TEEs, and requires no system software modifications.

REFERENCES

[1] Ittai Anati, Shay Gueron, Simon Johnson, and Vincent Scarlata. 2013. Innovative technology for CPU based attestation and sealing. In *Proceedings of the 2nd international workshop on hardware and architectural support for security and privacy*, Vol. 13. Citeseer, 7.

[2] Sergei Arnautov, Bohdan Trach, Franz Gregor, Thomas Knauth, André Martin, Christian Priebe, Joshua Lind, Divya Muthukumaran, Dan O'Keeffe, Mark Stillwell, David Goltzsche, David M. Eyers, Rüdiger Kapitza, Peter R. Pietzuch, and Christof Fetzer. 2016. SCONE: Secure Linux Containers with Intel SGX.. In *OSDI*. 689–703.

[3] Sumeet Bajaj and Radu Sion. 2014. TrustedDB: A Trusted Hardware-Based Database with Privacy and Data Confidentiality. *IEEE Trans. Knowl. Data Eng.* 26, 3 (2014), 752–765.

[4] Andrew Baumann, Marcus Peinado, and Galen C. Hunt. 2015. Shielding Applications from an Untrusted Cloud with Haven. *ACM Trans. Comput. Syst.* 33, 3 (2015), 8:1–8:26.

[5] Stefan Brenner, Colin Wulf, David Goltzsche, Nico Weichbrodt, Matthias Lorenz, Christof Fetzer, Peter R. Pietzuch, and Rüdiger Kapitza. 2016. SecureKeeper: Confidential ZooKeeper using Intel SGX.. In *MIDDLEWARE*. 14.

[6] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F. Wenisch, Yuval Yarom, and Raoul Strackx. 2018. Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution.. In *USS*. 991–1008.

[7] Chia che Tsai, Kumar Saurabh Arora, Nehal Bandi, Bhushan Jain, William Jannen, Jitin John, Harry A. Kalodner, Vrushali Kulkarni, Daniela Oliveira, and Donald E. Porter. 2014. Cooperation and security isolation of library OSes for multi-process applications.. In *EUROSYS*. 9:1–9:14.

[8] Victor Costan and Srinivas Devadas. 2016. Intel SGX Explained. *IACR Cryptol. ePrint Arch.* 2016 (2016), 86.

[9] Thomas Knauth, Michael Steiner, Somnath Chakrabarti, Li Lei, Cedric Xing, and Mona Vij. 2018. Integrating Remote Attestation with Transport Layer Security. *CoRR* abs/1801.05863 (2018).

[10] Dayeol Lee, David Kohlbrenner, Shweta Shinde, Krste Asanovic, and Dawn Song. 2020. Keystone: an open framework for architecting trusted execution environments.. In *EUROSYS*. 38:1–38:16.

[11] Joshua Lind, Christian Priebe, Divya Muthukumaran, Dan O'Keeffe, Pierre-Louis Aublin, Florian Kelbert, Tobias Reiher, David Goltzsche, David M. Eyers, Rüdiger Kapitza, Christof Fetzer, and Peter R. Pietzuch. 2017. Glamdring: Automatic Application Partitioning for Intel SGX.. In *USENIX ATC*. 285–298.

[12] Joshua Lund, Jeff Griffin, and Nolan Leake. 2019. *Technology preview for Secure Value Recovery.* https://signal.org/blog/secure-value-recovery/

[13] Sandro Pinto and Nuno Santos. 2019. Demystifying Arm TrustZone: A Comprehensive Survey. *ACM Comput. Surv.* 51, 6 (2019), 130:1–130:36.

[14] Christian Priebe, Kapil Vaswani, and Manuel Costa. 2018. EnclaveDB: A Secure Database Using SGX.. In *IEEE Symposium on Security and Privacy*. 264–278.

[15] Phillip Rogaway and Thomas Shrimpton. 2004. Cryptographic Hash-Function Basics: Definitions, Implications and Separations for Preimage Resistance, Second-Preimage Resistance, and Collision Resistance. *IACR Cryptol. ePrint Arch.* 2004 (2004), 35.

[16] AMD SEV-SNP. 2020. Strengthening VM isolation with integrity protection and more. *White Paper, January* (2020).