Challenges For Scaling Applications Across Enclaves

Jethro G. Beekman
Fortanix
jethro@fortanix.com

Donald E. Porter
Fortanix and UNC Chapel Hill
porter@fortanix.com

Abstract

At Fortanix, we are developing cloud-scale security infrastructure using SGX. For example, our Self-Defending Key Management Service (SDKMS) can span multiple machines and enclaves, rendering a more scalable and cost-effective alternative to a traditional Hardware Security Module (HSM).

This paper describes several subtle, practical, and under-explored problems in the space of building scalable, trusted applications, based on our experience building distributed SGX systems. In particular, we discuss shortcomings in remote attestation for microservice-style applications, software updates, and opportunities to reflect trustworthy development practices in attestation features.

CCS Concepts **• Security and privacy → Trusted computing**; **Distributed systems security**;

ACM Reference Format:
Jethro G. Beekman and Donald E. Porter. 2017. Challenges For Scaling Applications Across Enclaves. In *SysTEX'17: 2nd Workshop on System Software for Trusted Execution, October 28, 2017, Shanghai, China.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3152701.3152710

1 Introduction

Intel SGX creates an opportunity to build cost-scalable software systems that require trusted execution. A number of datacenter and cloud applications are designed to scale across relatively inexpensive servers, i.e., additional capacity can be added in small increments in monetary cost. Cloud computing unlocks even finer cost increments by sharing underutilized hardware. By extending commodity x86 chips with a trusted execution environment, SGX unlocks similar cost-scalability for security-sensitive applications.

Remote attestation is an essential feature for distributing trusted execution. In particular, for a simple client-server architecture, SGX's remote attestation is sufficient for a client and server to check that the other end of a network connection is running software that is signed by a trusted party. In other words, current remote attestation can establish trust between two points over a network.

The challenge for larger-scale applications is moving from point-to-point trust toward end-to-end trust. Modern cloud applications are often composed of multiple network-connected services, such as the increasingly popular microservices architecture. A client may connect to a front-end node, which then dispatches a request to a number of loosely coupled and potentially shared back-end services. For instance, a cloud-based document service may store the data in a back-end key-value store. For sensitive data, the client may wish to authenticate not only the front-end software, but also that the front-end is storing data in a trustworthy key-value back-end. In this example, the front-end is implicitly trusted to validate the back-end. As applications grow to span more enclaves, the TCB and attack surface also grow, increasing the risk that one enclave may become compromised and undermine the security of the entire system.

This paper also observes that a more expressive remote attestation mechanism could also be a natural point for fault isolation and end-to-end policy enforcement. For instance, with the right language and analysis tools (or sandbox), one can attest to certain security-relevant properties, such as the absence of additional network connections.

The observations in this paper are motivated by our experiences developing a cloud-scale key-management service using SGX. We note that these issues both apply to other trusted execution environments (TEEs) than SGX and are outside of the explicit threat model considered by most TEEs. On one hand, it is clear that rote use of remote attestation is not sufficient. On the other end of the spectrum there are very general-purpose solutions, such as BAN logic [2]. The challenge is identifying the right abstractions and principals which capture the essence of these problems, minimizing the risk of developer or administrator error. The paper also briefly discusses opportunities to integrate trustworthy development practices into the signing and attestation process.

2 End-To-End Attestation

For a composite service, remote attestation only reports on the front-end enclave. A client with particularly sensitive data may want end-to-end attestation reports, knowing that *all* binaries that handled client data were trusted, not simply the front-end. This issue becomes even more important

SysTEX '17, October 2017, Shanghai, China

ACM ISBN 978-1-4503-5097-6/17/10…$15.00
https://doi.org/10.1145/3152701.3152710

when a service handles data with varying regulatory compliance constraints, such as a cloud-based app handling some data for a medical clinic that is bound by HIPAA, as well as educational data that is subject to FERPA requirements.

We see an opportunity to develop more expressive attestation reports of composed services, wherein a client can see that only trusted binaries handled their data, end-to-end. Moreover, it would be desirable if these reports could attest that the back-end servers upheld certain policy goals, such as certified to be compliant with a given regulation.

In tension with this goal is the need to keep an architecture opaque to a client, if the architecture itself is a competitive advantage. Even in this situation, such a mechanism can be useful strictly within the application to detect compromised nodes. Additionally, one may be able to encode policies in a way that does not reveal system internals.

3 Software Updates

As bugs or vulnerabilities are discovered in the software, or the software simply improves, the software deployed at each node in a distributed system needs to be updated. It is often impractical to take all nodes offline at once, as many services need high degrees of availability; thus, rolling updates are common. Here we use the term “node” to refer to trusted software running in a remote enclave.

As an update is being deployed, the system needs a period where updated nodes still communicate with nodes that are running older (potentially vulnerable) software. However, after a reasonable period, if a node was missed during the update and remains on a stale version, it should be evicted from the system. Even a single trusted, yet compromised, node can have disastrous consequences.

Remote attestation alone cannot express such a policy. The vulnerable, older versions of the software are signed by a trusted party, and the new versions must initially trust the old. One possibility is to trust older versions of the software until a quorum of nodes have been updated, at which point the other nodes can be considered failed until they are updated. A challenge for this approach is carefully reasoning about the expected number of additional failures the system may encounter, lest the system become unavailable.

4 Trustworthy Development Practices

Attestation and software signing is only implicitly connected to the root issue of *trustworthiness*. There is a point at which a human being decides to sign a binary. In this section, we explain how this decision could be an explicit function of code quality efforts, as well as specific code attributes.

Most software companies already take a number of steps to ensure trustworthy code. In terms of code quality, current best practices are to develop features on a branch, and only merge that branch after it passes regression testing and code review. In terms of security, penetration tests or audits are common. Particularly-sensitive code may be analyzed for additional properties, or even formally verified.

However, a binary is ultimately signed by a human. This human may review the code or build process, but is unlikely to check the entire development history for best practices before signing. Moreover, the signer is making a number of tacit assumptions [5]. For instance, an internal attacker could have replaced the trusted compiler or resulting binary, or could have inserted a change deep in the version control history that was not properly reviewed.

We see an opportunity to reduce the risk of error and insider attacks by making binary signing an automated output of code quality tools, which could be configured to only sign a binary that was built using trusted tools, from code that followed trustworthy development practices.

We also see an opportunity for signing more expressive certificates: not just that a given binary was produced by a particular organization, but that the signer is certifying that the code has been tested (or is proved) to have certain properties. For instance, related to composition of microservices, one might sign that the code does not make any recursive calls to other microservices. A useful side-effect is the ability to determine if a vendor’s software is not trustworthy, such as finding a counter-example to an advertised property.

5 Related Work

A few prior works have investigated distributing applications across enclaves. Beekman et al. [1] describe how to share private data during software upgrades, but does not ensure a consistent secure state amongst all nodes. SGX has been integrated into data analytic frameworks [3, 6], but these primitives are unlikely to be generically reusable.

The Nexus OS includes an extensible, logical attestation framework [4], wherein applications can define rich policies, including policies dealing with time. A key difference is that Nexus is built on a trusted OS, which can provide a rich policy substrate; an open challenge is implementing Nexus-style attestations on top of SGX remote attestation primitives.

References

[1] Jethro G. Beekman, John L. Manferdelli, and David Wagner. 2016. Attestation Transparency: Building secure Internet services for legacy clients. In *AsiaCCS*.

[2] Michael Burrows, Martin Abadi, and Roger Needham. 1990. A Logic of Authentication. *TOCS* 8, 1 (Feb. 1990), 18–36.

[3] Felix Schuster, Manuel Costa, Cedric Fournet, Christos Gkantsidis, Marcus Peinado, Gloria Mainar-Ruiz, and Mark Russinovich. 2015. VC3: Trustworthy Data Analytics in the Cloud Using SGX. In *Oakland*.

[4] Emin Gün Sirer, Willem de Bruijn, Patrick Reynolds, Alan Shieh, Kevin Walsh, Dan Williams, and Fred B. Schneider. 2011. Logical Attestation: An Authorization Architecture for Trustworthy Computing. In *SOSP*.

[5] Ken Thompson. 1984. Reflections on Trusting Trust. *CACM* 27, 8 (Aug. 1984), 761–763.

[6] Wenting Zheng, Ankur Dave, Jethro G. Beekman, Raluca Ada Popa, Joseph E. Gonzalez, and Ion Stoica. 2017. Opaque: An Oblivious and Encrypted Distributed Analytics Platform. In *NSDI*.