# A Journey into Intel's SGX

Jessie Frazelle - QCon London

# A bit about Enclaves

# Intel SGX Explained

Victor Costan and Srinivas Devadas
victor@costan.us, devadas@mit.edu
*Computer Science and Artificial Intelligence Laboratory*
*Massachusetts Institute of Technology*

## ABSTRACT

Intel's Software Guard Extensions (SGX) is a set of extensions to the Intel architecture that aims to provide integrity and confidentiality guarantees to security-sensitive computation performed on a computer where all the privileged software (kernel, hypervisor, etc) is potentially malicious.

This paper analyzes Intel SGX, based on the 3 papers [14, 79, 139] that introduced it, on the Intel Software Developer's Manual [101] (which supersedes the SGX manuals [95, 99]), on an ISCA 2015 tutorial [103], and on two patents [110, 138]. We use the papers, reference manuals, and tutorial as primary data sources, and only draw on the patents to fill in missing information.

This paper does not reflect the information available in two papers [74, 109] that were published after the first version of this paper.

This paper's contributions are a summary of the Intel-specific architectural and micro-architectural details needed to understand SGX, a detailed and structured presentation of the publicly available information on SGX

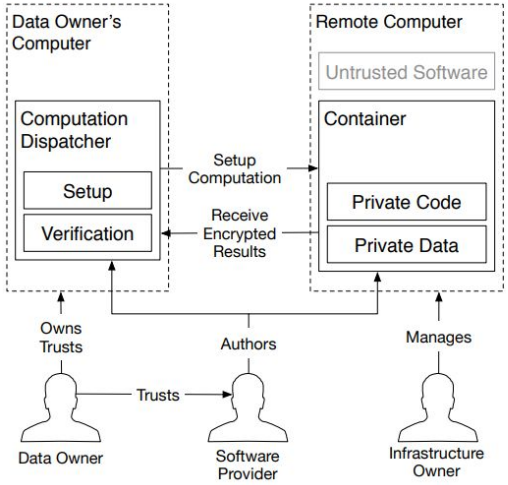



**Figure 1**: Secure remote computation. A user relies on a remote computer, owned by an untrusted party, to perform some computation on her data. The user has some assurance of the computation's integrity and confidentiality.

uploads the desired computation and data into the secure

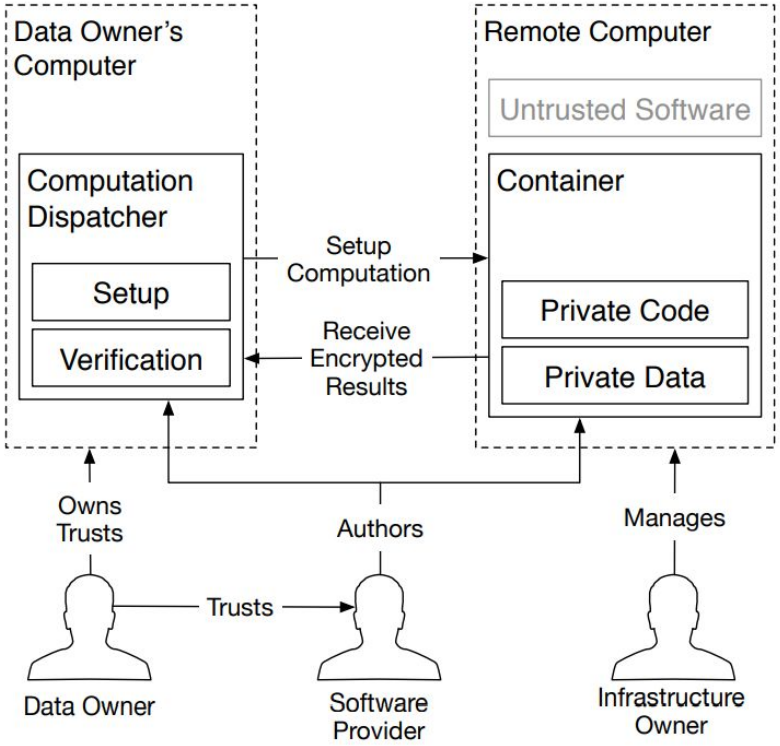



**Figure 1**: Secure remote computation. A user relies on a remote computer, owned by an untrusted party, to perform some computation on her data. The user has some assurance of the computation's integrity and confidentiality.

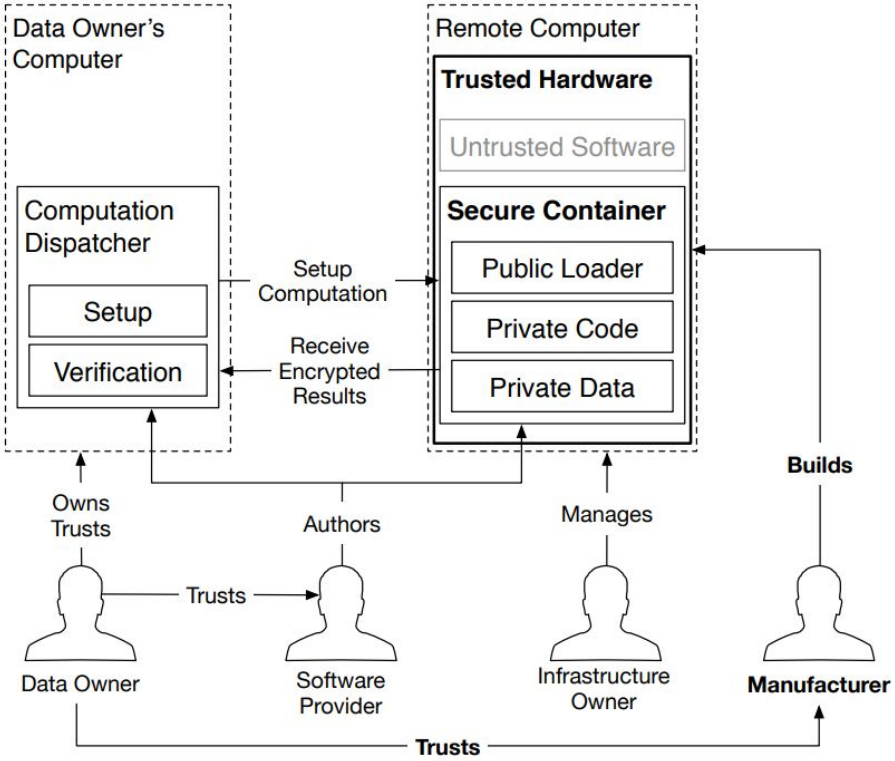
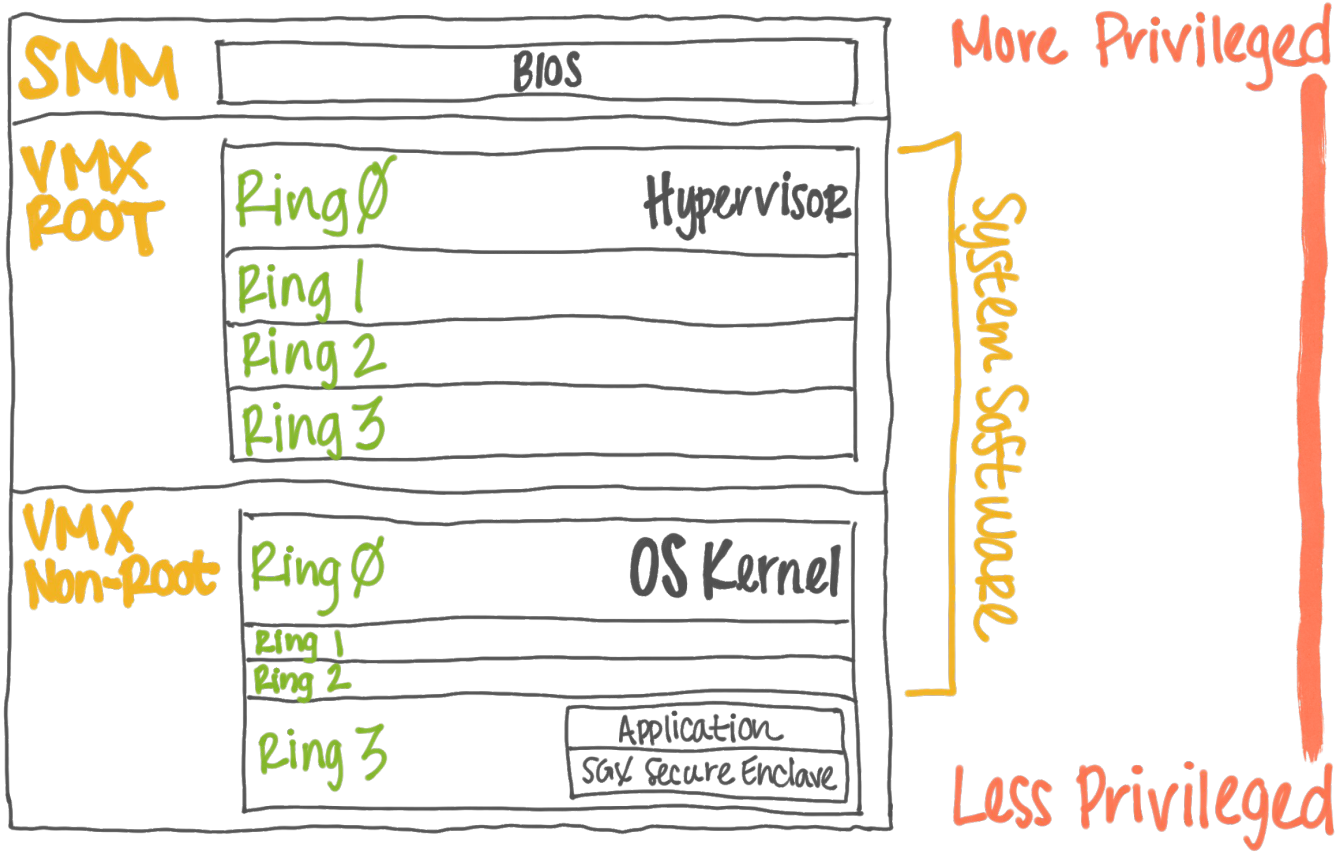


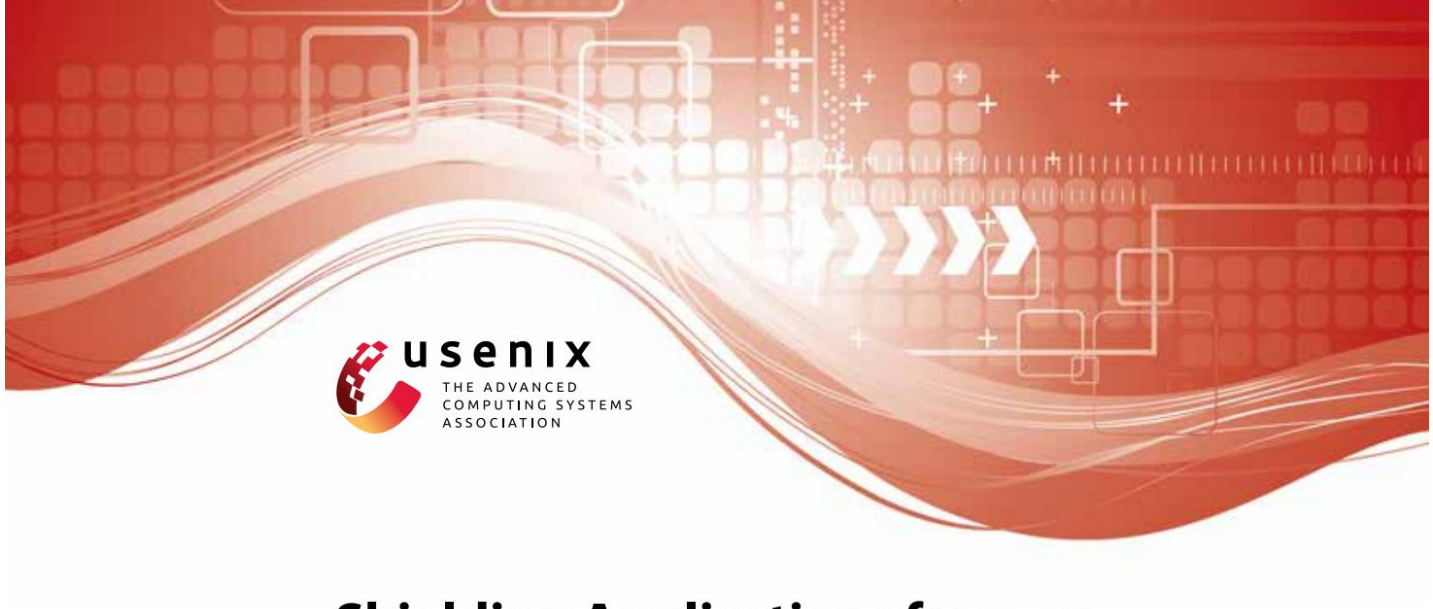

**Figure 2**: Trusted computing. The user trusts the manufacturer of a piece of hardware in the remote computer, and entrusts her data to a secure container hosted by the secure hardware.

# Software Privilege Levels



# How this all started

Originally meant for DRM




# Shielding Applications from an Untrusted Cloud with Haven

**Andrew Baumann, Marcus Peinado, and Galen Hunt,** ***Microsoft Research***

https://www.usenix.org/conference/osdi14/technical-sessions/presentation/baumann

**This paper is included in the Proceedings of the 11th USENIX Symposium on**


# SCONE: Secure Linux Containers with Intel SGX

Sergei Arnautov, Bohdan Trach, Franz Gregor, Thomas Knauth, and Andre Martin, *Technische Universität Dresden;* Christian Priebe, Joshua Lind, Divya Muthukumaran, Dan O'Keeffe, and Mark L Stillwell, *Imperial College London;* David Goltzsche, *Technische Universität Braunschweig;* Dave Eyers, *University of Otago;* Rüdiger Kapitza, *Technische Universität Braunschweig;* Peter Pietzuch, *Imperial College London;* Christof Fetzer, *Technische Universität Dresden*

https://www.usenix.org/conference/osdi16/technical-sessions/presentation/arnautov

# Challenges in the Design

# 1.

# Keep Code Small.

Keep the trusted computing base

SMALL

and with few dependencies

to lessen the risk of one having a

VULNERABILITY

it's kinda like...

if everything is

IN THE SANDBOX

then there

IS NO SANDBOX

# 2.
# Performance

Enclave thread

**MUST**

copy memory-based arguments

& **LEAVE**

the enclave

**BEFORE A SYSTEM CALL**

Memory pages for an

ENCLAVE

live in the

ENCLAVE PAGE CACHE

# AFTER CACHE MISS

Cache lines must be

# DECRYPTED

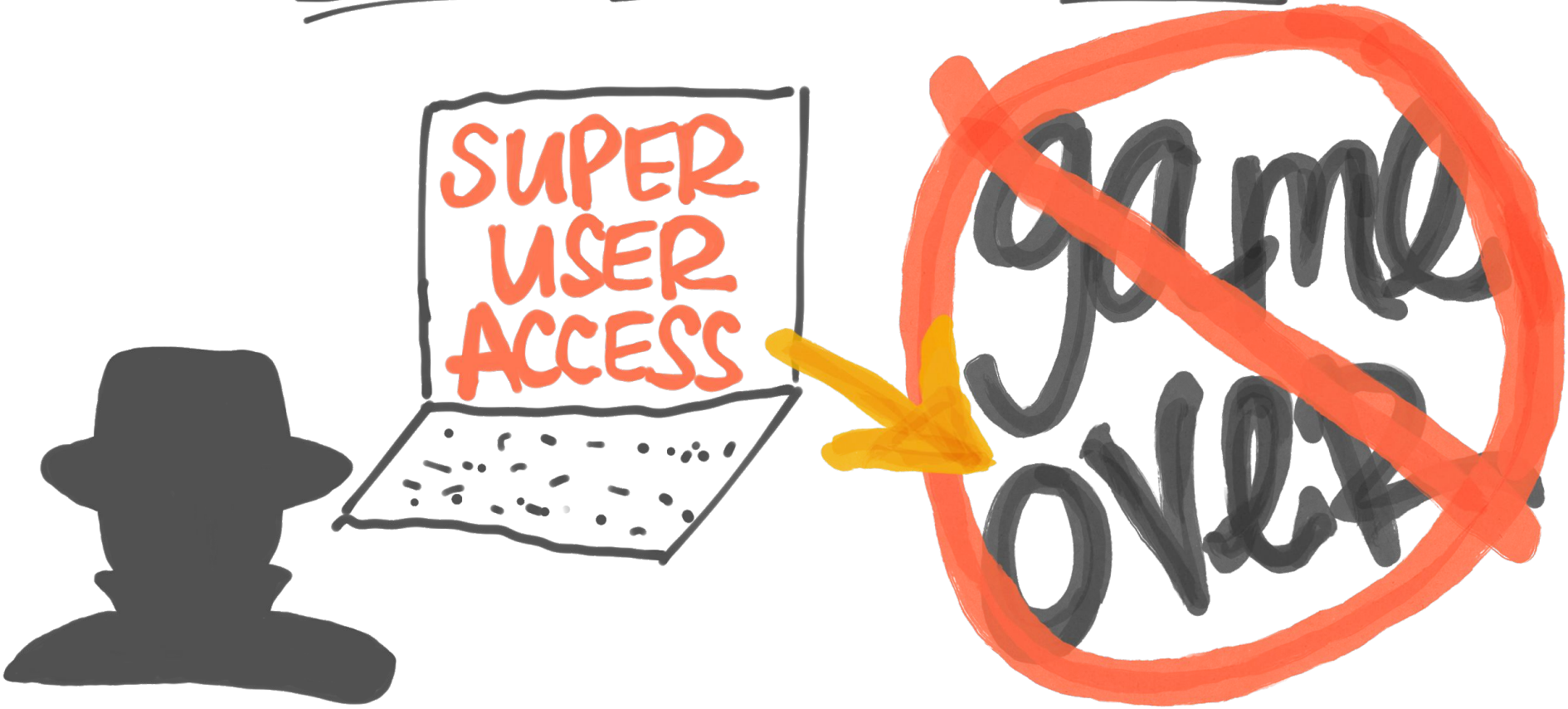
when fetched from

# MEMORY


Threat
Model

# Threat Model for Container Runtimes Today

# Threat Model for SCONE



# SCONE assumes...

# an attacker has/can

# SCONE Threat Model

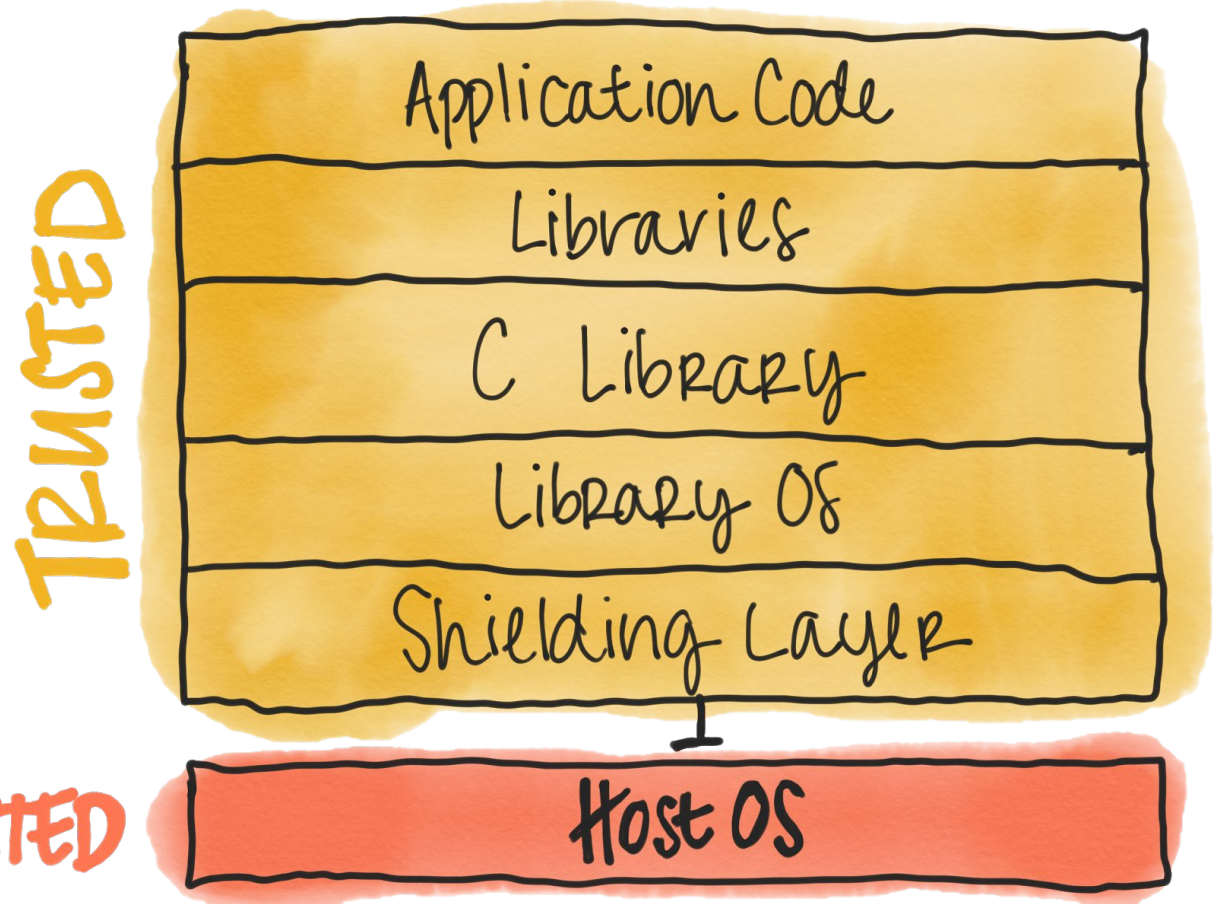
## DOES NOT COVER

Denial of Service Attacks

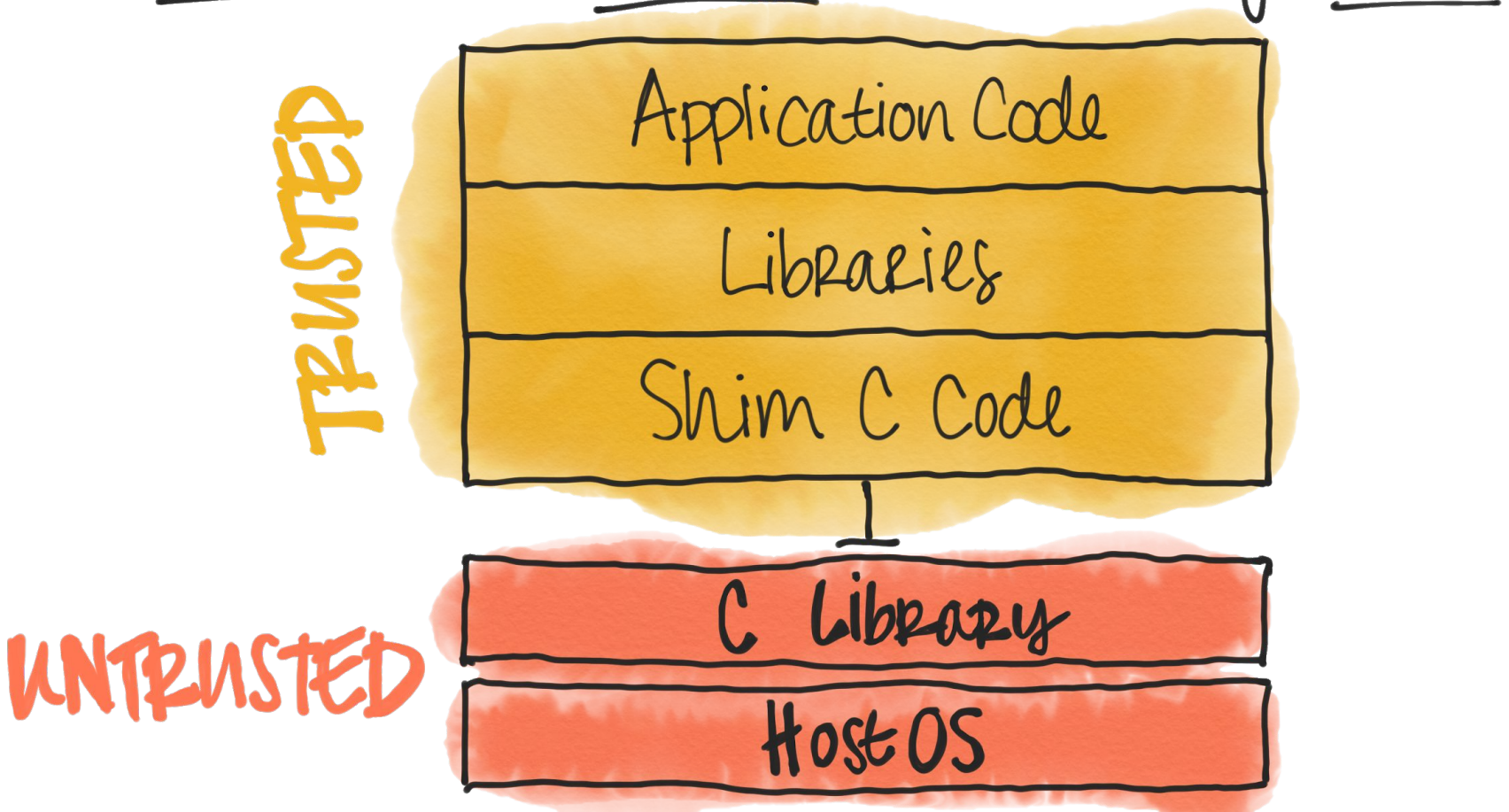
Side channel attacks over TIMING OR PAGE FAULTS

# Design Tradeoffs

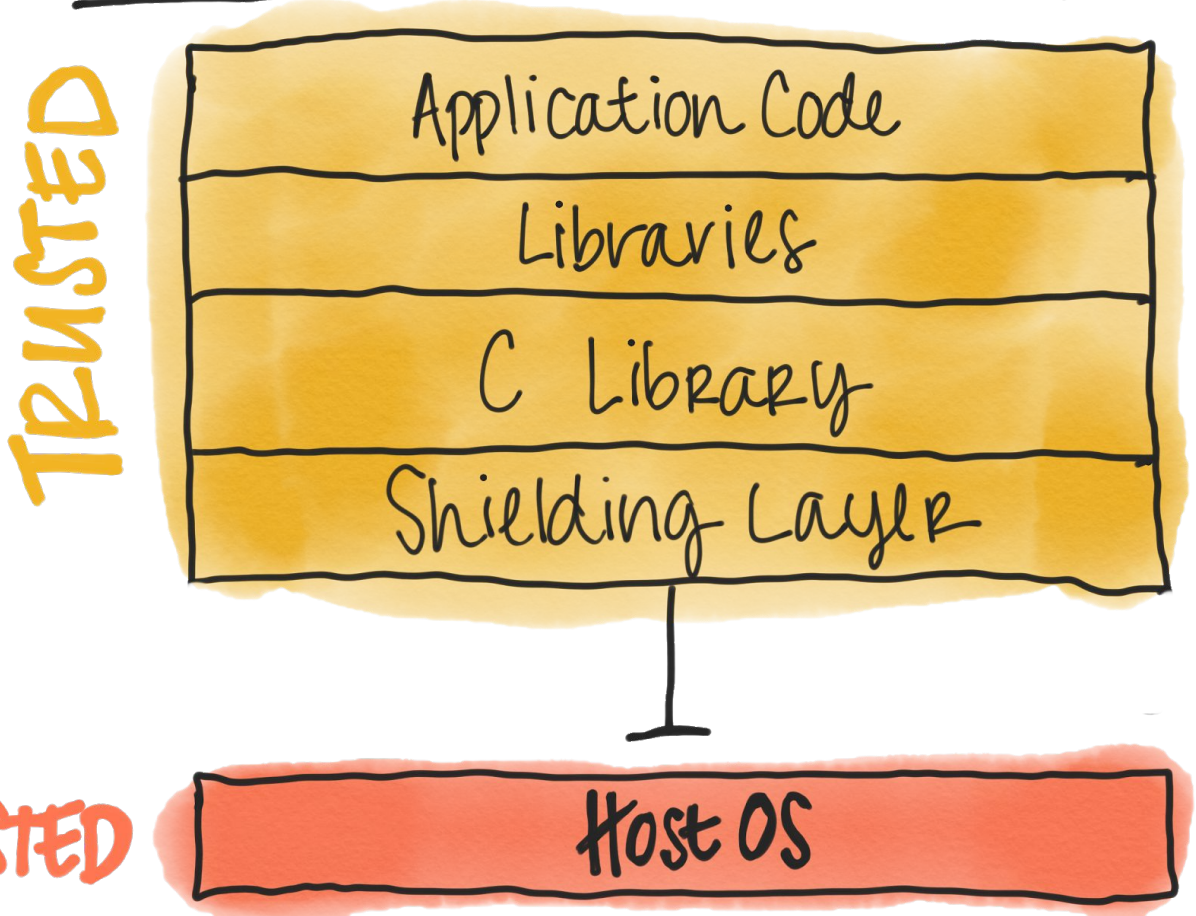
# Library Inside Trusted Computing Base



# Minimal Trusted Computing Base



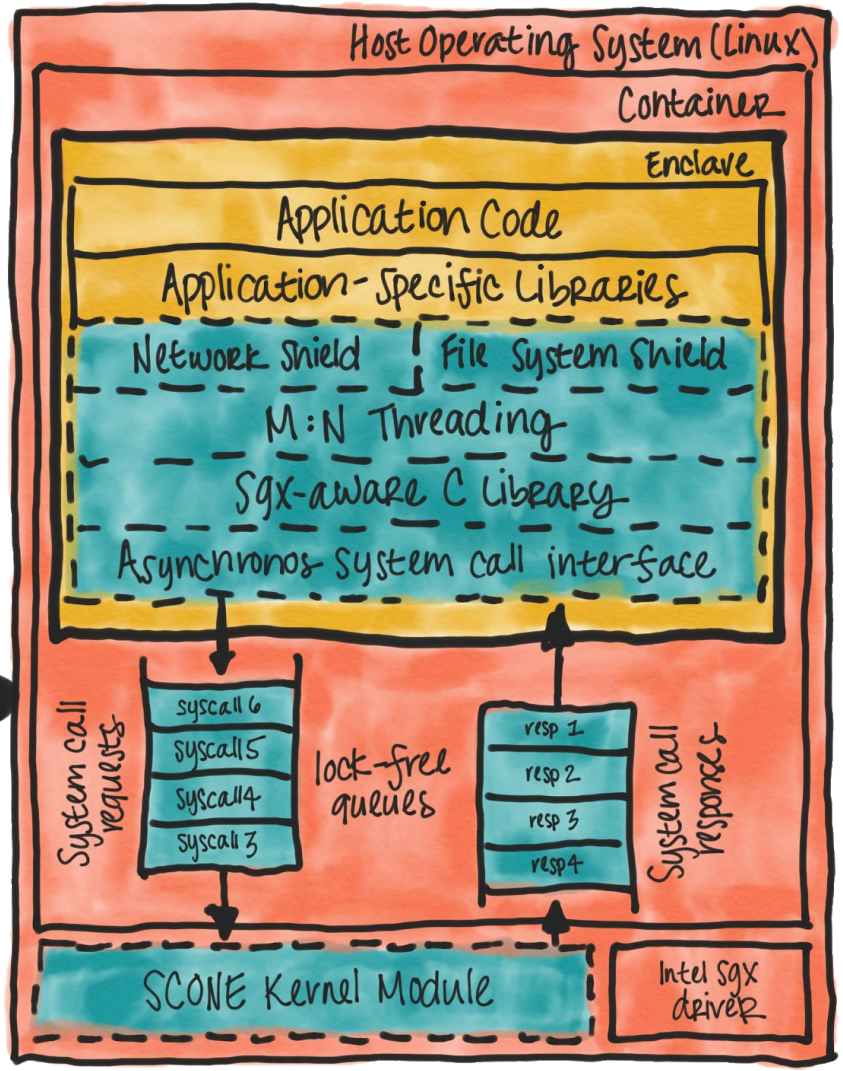
# Untrusted System Calls



System Calls being performed outside the enclave are expensive

Memory Page Faults have a Significant Overhead

# SCONE Design




TRUSTED

SCONE COMPONENTS

UNTRUSTED

# Further iterations on this

oscarlab / **graphene**

Watch 31 | Star 216 | Fork 92

Code | Issues 102 | Pull requests 78 | Projects 0 | Wiki | Insights

Graphene / Graphene-SGX Library OS - a library OS for Linux multi-process applications, with Intel SGX support
https://github.com/oscarlab/graphene/…

linux | compatibility | virtualization | sgx

**439** commits | **18** branches | **5** releases | **21** contributors | LGPL-3.0

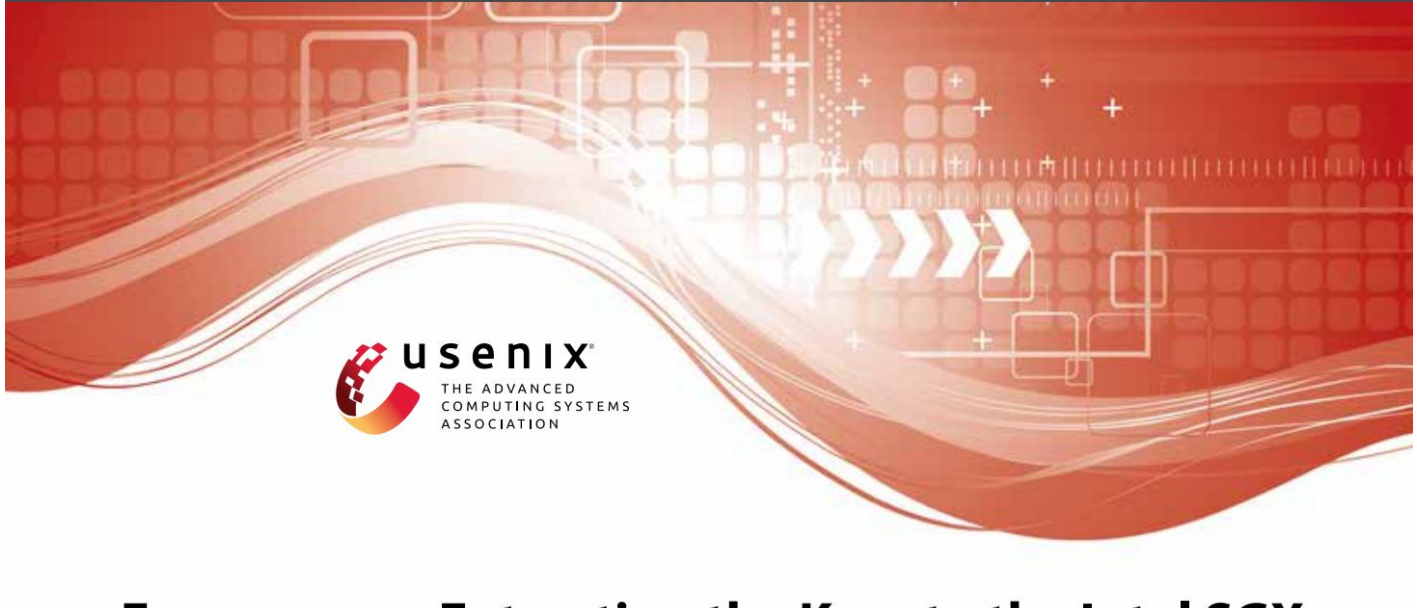
Branch: **master** | **New pull request** | **Create new file** | **Upload files** | **Find file** | **Clone or download**

**bigdata-memory** and **donporter** Fix null test and write test issues about the stat parameter ... Latest commit `5d799d2` on Oct 18, 2018

| | | |
|---|---|---|
| .github | [Docs] Simplify our PR template | 27 days ago |
| Jenkinsfiles | [Jenkinsfiles] Optimize the dockerfile for Ubuntu | 14 days ago |
| LibOS | Fix null test and write test issues about the stat parameter | 5 days ago |
| Pal | [Pal/Linux-SGX] Fix typo: externel -> external | 7 days ago |
| Runtime | Fix Bash shebangs compatibility | 27 days ago |
| Scripts | Fix Bash shebangs compatibility | 27 days ago |
| Tools | Tools/gsce: Clean up and optimize the integration of docker container… | 6 days ago |
| .clang-format | First cut at a .clang-format for the project | 3 months ago |

# The weird thing about Launch Control

# Attacks on SGX




# FORESHADOW: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution

Jo Van Bulck, *imec-DistriNet, KU Leuven;* Marina Minkin, *Technion;* Ofir Weisse, Daniel Genkin, and Baris Kasikci, *University of Michigan;* Frank Piessens, *imec-DistriNet, KU Leuven;* Mark Silberstein, *Technion;* Thomas F. Wenisch, *University of Michigan;* Yuval Yarom, *University of Adelaide and Data61;* Raoul Strackx, *imec-DistriNet, KU Leuven*

https://www.usenix.org/conference/usenixsecurity18/presentation/bulck

This paper is included in the Proceedings of the
27th USENIX Security Symposium

# Foreshadow-NG: Breaking the Virtual Memory Abstraction with Transient Out-of-Order Execution

**Revision 1.0 (August 14, 2018)**

Ofir Weisse[3], Jo Van Bulck[1], Marina Minkin[2], Daniel Genkin[3], Baris Kasikci[3], Frank Piessens[1], Mark Silberstein[2], Raoul Strackx[1], Thomas F. Wenisch[3], and Yuval Yarom[4]

[1]*imec-DistriNet, KU Leuven,* [2]*Technion,* [3]*University of Michigan,* [4]*University of Adelaide and Data61*

## Abstract

In January 2018, we discovered the *Foreshadow* transient execution attack (USENIX Security'18) targeting Intel SGX technology. Intel's subsequent investigation of our attack uncovered two closely related variants, which we collectively call *Foreshadow-NG* and which Intel refers to as L1 Terminal Fault. Current analyses focus mostly on mitigation strategies, providing only limited insight into the attacks themselves and their consequences. The aim of this report is to alleviate this situation by thoroughly analyzing Foreshadow-type attacks and their implications in the light of the emerging transient execution research area.

At a high level, whereas previous generation Meltdown-type attacks are limited to reading privileged supervisor data *within* the attacker's virtual address space, Foreshadow-NG attacks completely bypass the

tion requires different computational tasks belonging to separate security domains to be isolated from each other and prevented from reading each other's memory. In modern computer architectures this is typically achieved via hardware-backed virtual memory, where each process has its own separate virtual address space. When a process accesses some memory location in its virtual address space, the hardware translates the location's address into the corresponding physical address. Beyond the convenience of simulating a memory space much larger then the system's physical memory and the avoidance of address collisions across virtual address spaces, virtual memory serves as an effective security mechanism. Specifically, because addresses used by a process are always translated using the hardware-based translation mechanism, on a correctly functioning hardware, a process cannot "name" physical addresses belonging to

# Malware Guard Extension: Using SGX to Conceal Cache Attacks (Extended Version)

Michael Schwarz
Graz University of Technology
Email: michael.schwarz@iaik.tugraz.at

Samuel Weiser
Graz University of Technology
Email: samuel.weiser@iaik.tugraz.at

Daniel Gruss
Graz University of Technology
Email: daniel.gruss@iaik.tugraz.at

Clémentine Maurice
Graz University of Technology
Email: clementine.maurice@iaik.tugraz.at

Stefan Mangard
Graz University of Technology
Email: stefan.mangard@iaik.tugraz.at

***Abstract*—In modern computer systems, user processes are isolated from each other by the operating system and the hardware. Additionally, in a cloud scenario it is crucial that the hypervisor isolates tenants from other tenants that are co-located on the same physical machine. However, the hypervisor does not protect tenants against the cloud provider and thus the supplied operating system and hardware. Intel SGX provides a mechanism that addresses this scenario. It aims at protecting user-level software from attacks from other processes, the operating system, and even physical attackers.**

**In this paper, we demonstrate fine-grained software-based side-channel attacks from a malicious SGX enclave targeting co-located enclaves. Our attack is the first malware running on real SGX hardware, abusing SGX protection features to conceal itself. Furthermore, we demonstrate our attack both in a native environment and across multiple Docker containers. We perform a *Prime+Probe* cache side-channel attack on a co-located SGX enclave running an up-to-date RSA implementation**

attacks can recover cryptographic secrets, such as AES [2], [3] and RSA [4] keys, across virtual machine boundaries.

Intel introduced a new hardware extension SGX (Software Guard Extensions) [5] in their CPUs, starting with the Skylake microarchitecture. SGX is an isolation mechanism, aiming at protecting code and data from modification or disclosure even if all privileged software is malicious [6]. This protection uses special execution environments, so-called enclaves, which work on memory areas that are isolated from the operating system by the hardware. The memory area used by the enclaves is encrypted to protect the application's secrets from hardware attackers. Typical use cases include password input, password managers, and cryptographic operations. Intel recommends storing cryptographic keys inside enclaves and claims that side-channel attacks "are thwarted since the memory is protected by hardware encryption" [7].

# CacheZoom: How SGX Amplifies The Power of Cache Attacks

Ahmad Moghimi
*Worcester Polytechnic Institute*
*amoghimi@wpi.edu*

Gorka Irazoqui
*Worcester Polytechnic Institute*
*girazoki@wpi.edu*

Thomas Eisenbarth
*Worcester Polytechnic Institute*
*teisenbarth@wpi.edu*

## Abstract

In modern computing environments, hardware resources are commonly shared, and parallel computation is widely used. Parallel tasks can cause privacy and security problems if proper isolation is not enforced. Intel proposed SGX to create a trusted execution environment within the processor. SGX relies on the hardware, and claims runtime protection even if the OS and other software components are malicious. However, SGX disregards side-channel attacks. We introduce a powerful cache side-channel attack that provides system adversaries a high resolution channel. Our attack tool named *CacheZoom* is able to virtually track all memory accesses of SGX enclaves with high spatial and temporal precision. As proof of concept, we demonstrate AES key recovery attacks on commonly used implementations including those that were believed to be resistant in previous scenarios. Our

the operating system (OS) provides security and privacy services. In cloud computing, cloud providers and the hypervisor also become part of the Trusted Computing Base (TCB). Due to the high complexity and various attack surfaces in modern computing systems, keeping an entire system secure is usually unrealistic [19, 33].

One way to reduce the TCB is to outsource security-critical services to Secure Elements (SE), a separate trusted hardware which usually undergoes rigorous auditing. Trusted Platform Modules (TPM), for example, provide services such as cryptography, secure boot, sealing data and attestation beyond the authority of the OS [40]. However, SEs come with their own drawbacks: they are static components and connected to the CPU over an untrusted bus. Trusted Execution Environments (TEE) are an alternative, which provide similar services within the CPU. A TEE is an isolated environment to

# Practical Enclave Malware with Intel SGX

Michael Schwarz, Samuel Weiser, Daniel Gruss

Graz University of Technology

**Abstract.** Modern CPU architectures offer strong isolation guarantees towards user applications in the form of enclaves. For instance, Intel's threat model for SGX assumes fully trusted enclaves, yet there is an ongoing debate on whether this threat model is realistic. In particular, it is unclear to what extent enclave malware could harm a system. In this work, we practically demonstrate the first enclave malware which fully and stealthily impersonates its host application. Together with poorly-deployed application isolation on personal computers, such malware can not only steal or encrypt documents for extortion, but also act on the user's behalf, e.g., sending phishing emails or mounting denial-of-service attacks. Our SGX-ROP attack uses new TSX-based memory-disclosure primitive and a write-anything-anywhere primitive to construct a code-reuse attack from within an enclave which is then inadvertently executed by the host application. With SGX-ROP, we bypass ASLR, stack canaries, and address sanitizer. We demonstrate that instead of protecting users from harm, SGX currently poses a security threat, facilitating so-called super-malware with ready-to-hit exploits. With our results, we seek to demystify the enclave malware threat and lay solid ground for future research on and defense against enclave malware.

**Keywords:** Intel SGX, Trusted Execution Environments, Malware