SysTEX'18
Toronto, Ontario, Canada

TEEshift: Protecting Code Confidentiality by Selectively Shifting Functions into TEEs

Titouan Lazard*, Johannes Götzfried†, Tilo Müller†,
Gianni Santinelli‡, and Vincent Lefebvre‡

*University of Rennes 1
ENS Rennes, France

†Department of Computer Science
FAU Erlangen-Nuremberg, Germany

‡Tages SAS Solidshield
France

October 15, 2018

Trusted Computing Architectures

General Purpose Hardware

- *Static Root-of-Trust* (SRoT)
  - *Trusted Platform Module* (TPM)
  - ARM TrustZone
- *Dynamic Root-of-Trust* (DRoT)
  - Intel SGX
  - AMD SME/SEV
  - Intel TME/MKTME

Custom Hardware

- Soteria / Sancus
- Atlas
- Sofia
- etc.

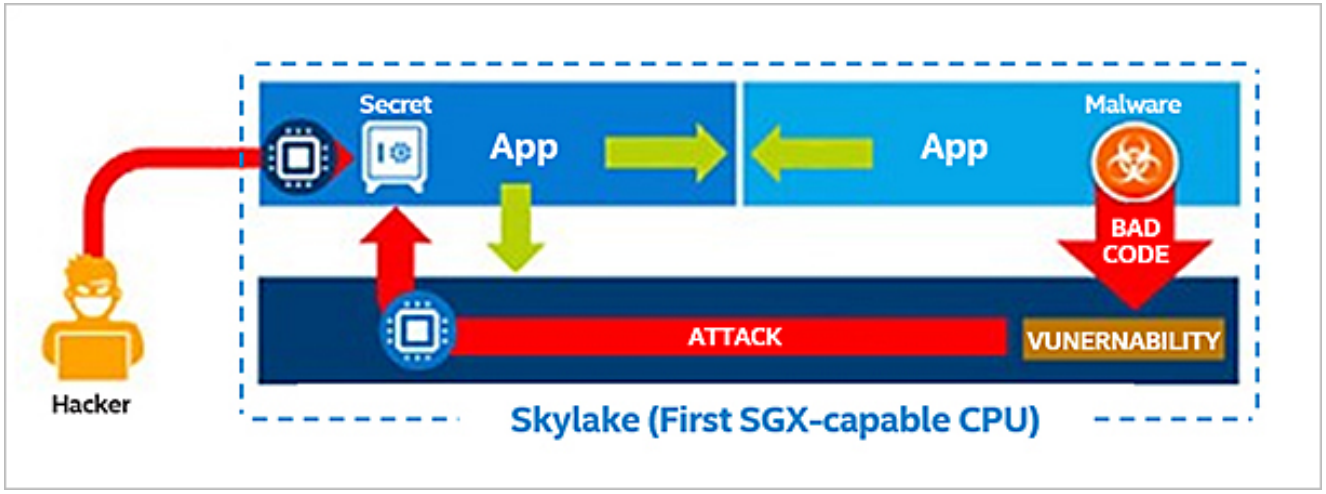
TrustZone®
System Security by ARM

Intel SGX

- Architecture guaranteeing confidentiality *and* integrity of enclaves
- Enclaves are protected against higher privileged software
  $\rightarrow$ including operating system and potentially hypervisor
- Enclaves can be started dynamically at any time
  $\rightarrow$ Dynamic Root of Trust (DRoT)
- Remote Attestation is supported via Intel's Attestation Services (IAS)

Enclaves live within existing process address spaces

- memory is managed by the (untrusted) operating system
- obey usual address translation rules
- can access memory of their host process



AMD SME/SEV

Architecture guaranteeing confidentiality of hypervisor and VMs

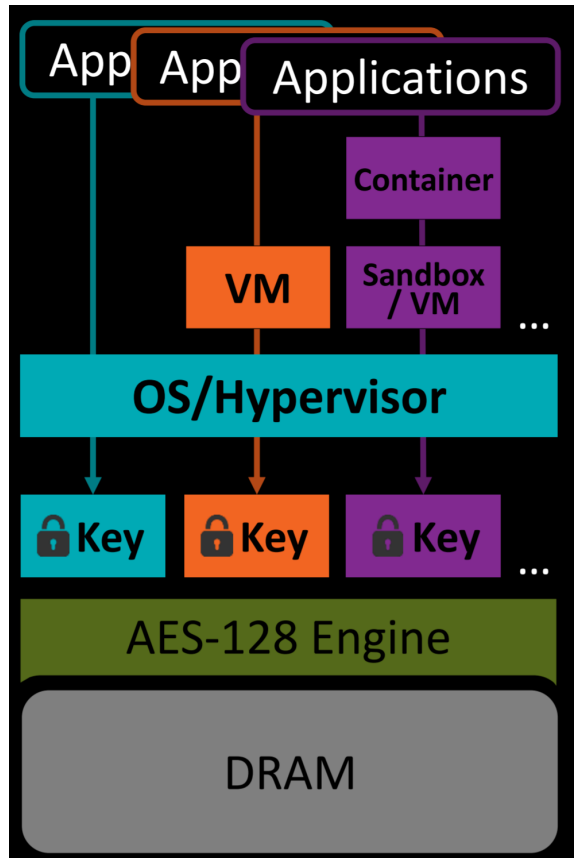
- Advanced memory encryption engine
- Keys are set by the hypervisor
  $\rightarrow$ resetting of keys will result in garbage
- Remote attestation during VM initialization

Entirely different architecture compared to SGX

- motivated by untrusted cloud scenario
- VMs typically results in huge TCB
- even more exposed to side channels

Intel tries to copy this concept with TME/MKTME

- Only confidentiality, no remote attestation
- Not bound to virtualization extensions
- Key identifier inside top-most bits of the virtual address

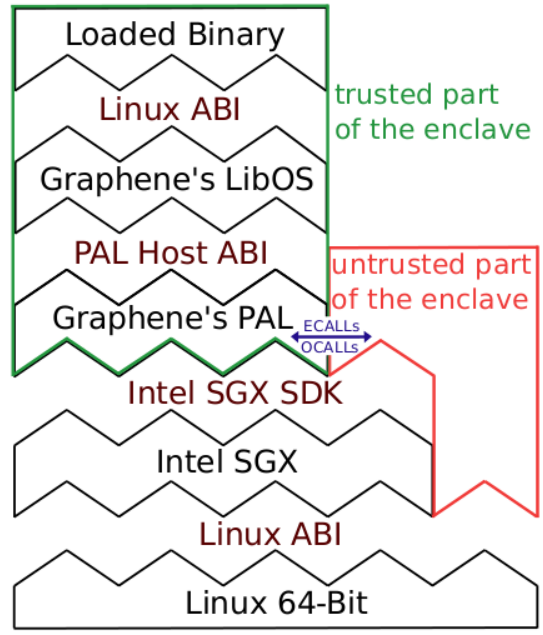


Idea: Protect Existing Code with TEEs

- Overcome major limitation: the need for code rewriting
- Existing TEEs are fundamentally different in their design
- Goal: Protect existing code without modifications across TEEs

Idea is really old at our lab

- Intel was faster: Graphene-SGX
- Google is faster: Asylo
- Only considered “esoteric” use cases so far (e.g. isolating kernel components)



Cloudshift

Tages SAS Solidshield

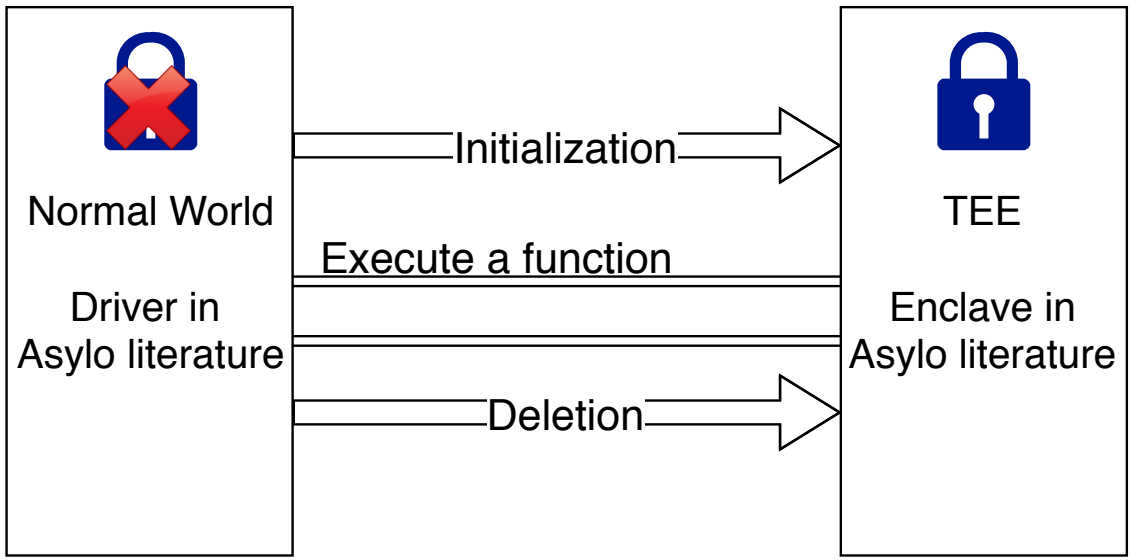
- French company for software protection and code obfuscation
- Code Virtualization (like VMprotect)
- Code Remotization (a.k.a. CloudShift)
- Anti-Tampering, Device Binding, Licensing, and more

From Cloudshift to TEEshift

- No Internet connectivity required
- Binary rewriting based
- Select functions worth protecting rather than the whole binary

TEEshift Overview

- A post-processing ELF-rewriting command line tool
- Transparently shifts user-selected functions into TEEs
- Builds on top of a new Google project Asylo



Asylo currently only supports SGX simulation backend

- Plans to support AMD SEV (and maybe even TrustZone) soon
- We are currently developing our own SEV backend

TEEshift Workflow

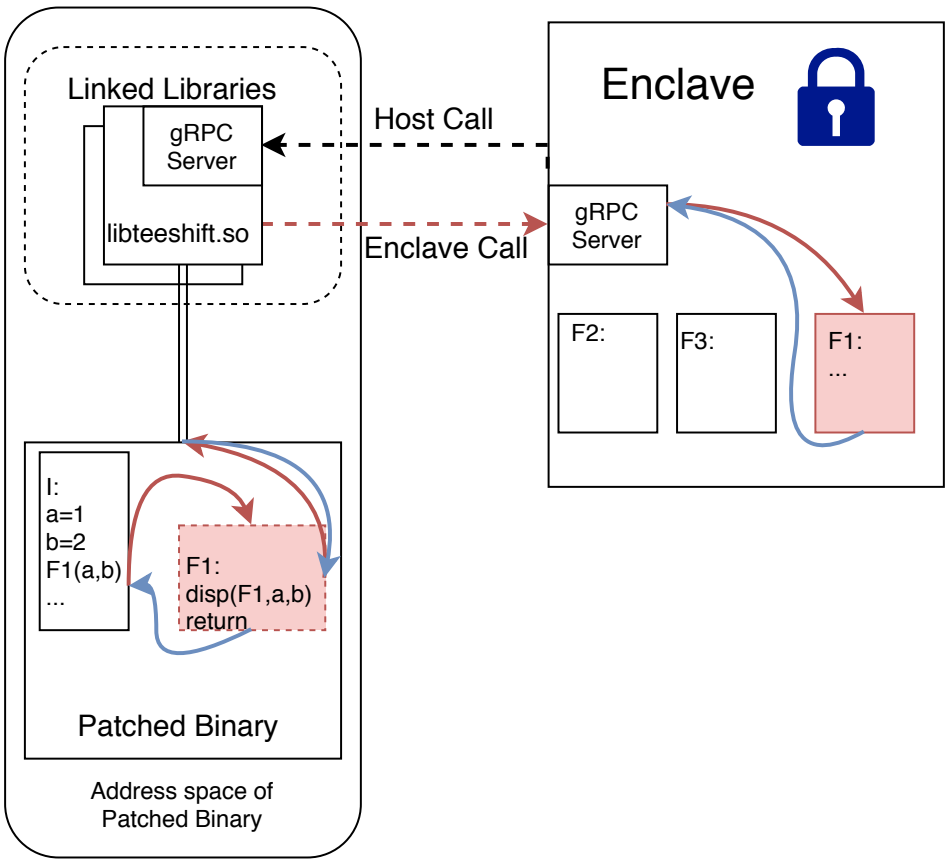
Mostly a DRM solution (only code confidentiality is supported)

- To be used as a drop-in replacement for Cloudshift

1. User-selected target functions are statically analysed
2. Pair of Asylo driver and enclave is built with modified function code
3. Asylo driver is compiled as a shared library
4. Target binary is patched to redirect function calls to the dispatcher within the Asylo library
5. Original functions are stripped from the binary

TEEshift Architecture

- Multithreaded architecture using gRPC for communication
- Asylo is compiled as a shared library and loaded into the original (patched) binary



TEEshift Challenges

Managing calls from and to the enclave

- Shared library loading into patched binary through LD_PRELOAD
- Control flow redirection to dispatcher
  - use `r11` register to identify caller
  - always pass 5 arguments (`RDI` to `r8`)
- Callbacks (non-enclave calls) are replaced by calls to gRPC client running inside TEE thread

Pointer dereferencing

- Protected function can contain arbitrary dereferecements
- Cannot be handled directly
- Instead special function is called which does the read/write

$\rightarrow$ High Overhead

TEEshift Challenges (cont.)

Current limitation: Accessing TEE memory

- Original binary is not allowed to access TEE memory
- Also does not work because of multithreaded setup
- Cloudshift suffers from the same limitation

$\rightarrow$ We do not consider this a big limitation (security critical anyway)

TEEshift Evaluation

Evaluation of TEEshift using a real world game: Teeworlds (you get it?)

- No real overhead noticeable if connection functions are protected
- However, overhead can get arbitrarily bad depending on the choice

Lab Setup

- Call to enclave function adds approximately 0.5ms
- Protected `Add()` function of the `ServerBrowser` object
- 812 calls to `Add()` did not affect interactivity

→ Initialisation functions are a good target
(trade-off between security and performance)

Future Work

Consider data confidentiality in addition to code confidentiality

- Not trivial for shared or global data
- Static analysis needed

Automatic selection of functions to shift

- Currently developer has to select functions to protect
  (given on the command line)
- Select functions which are worthwhile being protected but rarely used
- Rely on combination of static analysis and code complexity metrics

Conclusion

TEEshift as drop-in replacement for Cloudshift

- Works across multiple TEEs (Asylo backends)
- Primarily designed to protect code confidentiality
- Works directly on the binary (user-friendly)

$\rightarrow$ *Best-effort* security as alternative to obfuscation

Questions?

42.