SysTEX'16
Trento, Italy

Isolating Operating System Components with Intel SGX

Lars Richter, Johannes Götzfried, Tilo Müller

Department of Computer Science
FAU Erlangen-Nuremberg, Germany

December 12, 2016

Motivation or *Why are we here?*

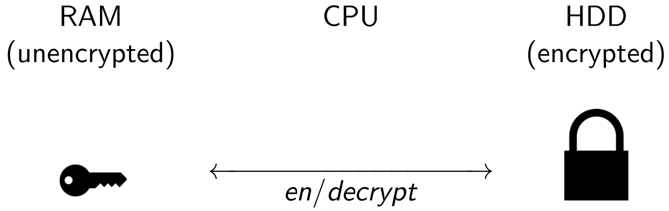
When SGX was released we thought of what to do with it

- Cloud-related solutions more or less addressed already
  - Haven and VC3 (at least on a conceptual level)
- Attacks on SGX (maybe yet to come)
- Using SGX as intended
  - not very promising in academia

What would be conceptually new and could benefit from SGX?
$\rightarrow$ Protecting kernel components with SGX

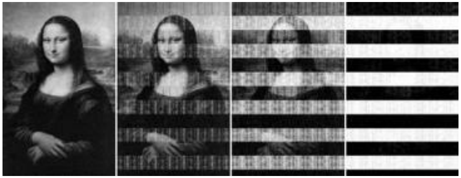
Our Background: CPU-bound Disk Encryption Schemes

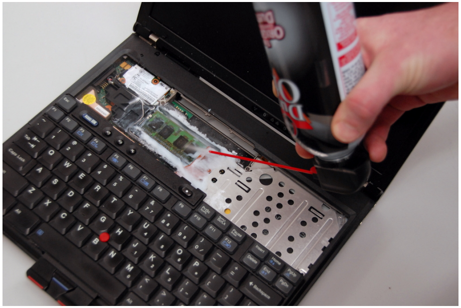
State-of-the-Art (software-based) FDE solutions do *not* protect data effectively if an adversary gains *physical access*!



Coldboot Attack

Disk Encryption Key in RAM
$\rightarrow$ Exploit remanence effect of RAM





Our Background: CPU-bound Disk Encryption Schemes

CPU-bound disk encryption schemes prevent coldboot attacks

TRESOR [USENIX'11], TreVisor [ACNS'12], MARK [TISSEC'14]

- Use the x86 debug registers *dr0* to *dr3* as key storage
- Utilize SSE registers to execute the AES algorithm
- Key schedule is calculated on-the-fly

We also implemented a solution for ARM: ARMORED [ARES'13].

→ SGX could help overcoming some limitations of CPU-bound encryption!

Why Protecting Kernel Components in General?

Today's kernels have huge code bases

- Kernel bugs are still common (cf. DirtyCow)
- The largest parts of those kernels are device drivers
  - Potentially developed from 3rd parties
- Not all parts can be equally trusted

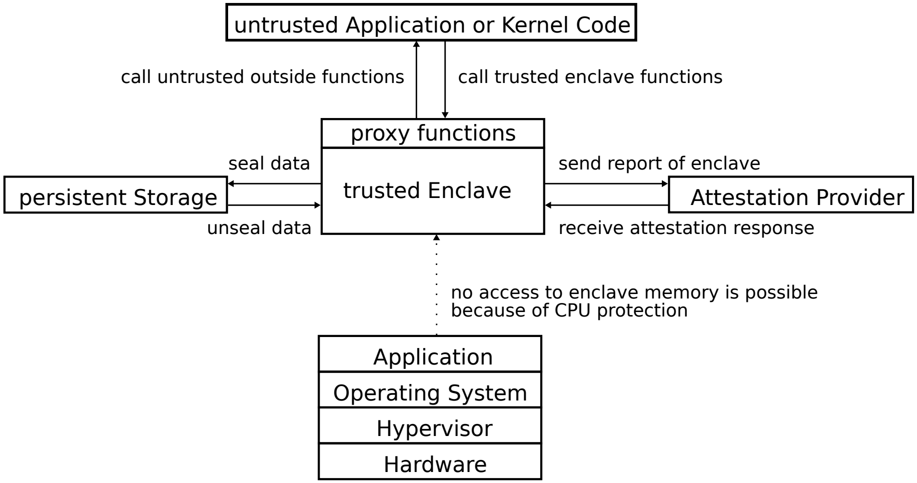
Shielding kernel components with SGX

- Only one component, e.g., a driver, is compromised
- Other parts like scheduler or disk encryption are still protected

Challenges

- Limited set of allowed instructions
- Fixed layout and memory mappings after initialization
- Enclaves can only be started through Intel's LE

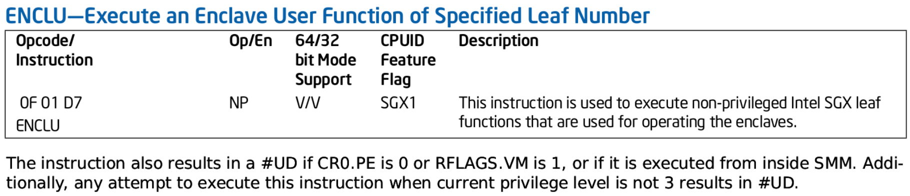
How We Imagined SGX to Work


untrusted Application or Kernel Code
call untrusted outside functions
call trusted enclave functions
proxy functions
trusted Enclave
seal data
unseal data
persistent Storage
send report of enclave
receive attestation response
Attestation Provider
no access to enclave memory is possible because of CPU protection
Application
Operating System
Hypervisor
Hardware

There Was One Problem However...

We didn't follow one very basic rule!

RTFM



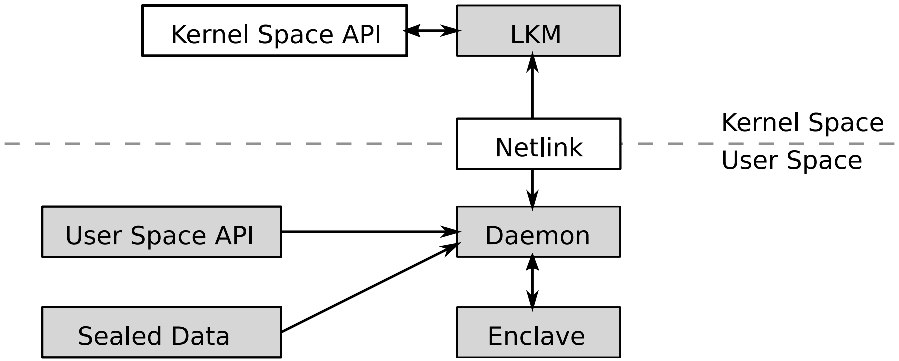
ENCLU—Execute an Enclave User Function of Specified Leaf Number

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
0F 01 D7 ENCLU	NP	V/V	SGX1	This instruction is used to execute non-privileged Intel SGX leaf functions that are used for operating the enclaves.

The instruction also results in a #UD if CR0.PE is 0 or RFLAGS.VM is 1, or if it is executed from inside SMM. Additionally, any attempt to execute this instruction when current privilege level is not 3 results in #UD.

Shielding Kernel Components

Entering an enclave in ring zero is not possible
$\rightarrow$ We decided to move kernel functionality to user mode first



Shielding Kernel Components

Netlink interfaces are used for communication

- Between the LKM and the daemon
- No kernel patching is required
- Usable through callback messages by kernel and user space

Loadable kernel module

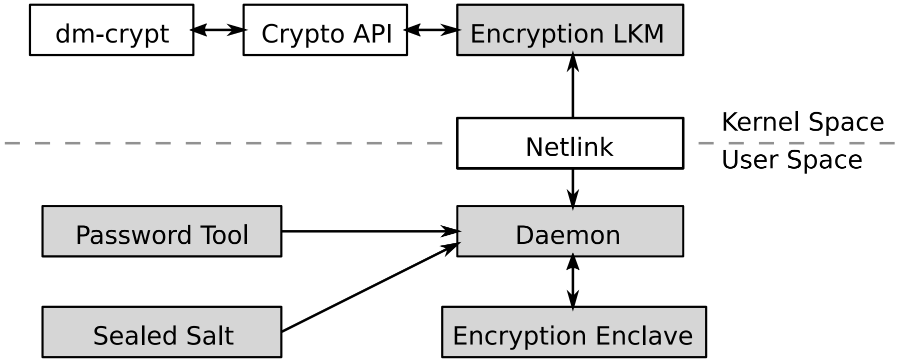
- Everything on the kernel side is implemented solely within an LKM
- The LKM can use any kernel API available
- Passes certain requests to the daemon

Daemon is able to offer additional functionality

- Relays most requests to the enclave
- Interacts with user space API
- Stores sealed data

Shielding Linux Full Disk Encryption (FDE)

Prototype Implementation to protect one specific kernel functionality



Protection against Physical Attacks

Typical FDE solutions

- Protect the encryption key only by *logical* means
- By relying on isolation and separation between kernel and user space
- Passphrase and derived key are usually stored in main memory

SGX-kernel solution

- Protects against physical attacks on main memory
- In particular against cold-boot and DMA attacks
- Additional salt prevents derivation of key solely from passphrase

$\rightarrow$ Practically stronger than CPU-bound encryption schemes

Workflow

SGX-kernel FDE initialization sequence

1. Load the LKM and start the daemon
2. Set password for FDE using the daemon (to avoid key leaking)
3. Derive encryption key using passphrase and sealed salt (PBKDF2)
4. Establish Netlink connection between LKM and daemon

SGX-kernel FDE Data Encryption and Decryption

1. The LKM registers a cipher within the Linux crypto API (currently only AES is offered by the encryption enclave)
2. Netlink callbacks are used on encryption and decryption for each block
3. The encryption or decryption request is processed by the enclave

Performance

Test System

- Dell Inspiron 7559 running Ubuntu 15.10 (kernel version 4.4.7)
- Intel i7-6700HQ CPU
- 16 gigabytes of main memory
- Seagate ST1000LM024 hard drive

Performance Results for SGX-kernel FDE with AES (reading and writing speeds in MB/s)

Test	Plain	AES	SGX
dd 100mb block write	107.0	104.5	1.1
hdparm uncached read	110.1	113.7	1.1
hdparm cached read	13,289.5	12,004.3	1,576.7

Security

Security guarantees of SGX-kernel FDE

- All guarantees inherited from SGX enclaves in general such as protection against tampering with the kernel component
- The disk encryption key is always kept within the enclave and never exposed to the outside world

The following components need to be obtained for a successful attack

- user password
- sealed salt (possibly stored on thumb drive)
- unmodified enclave which sealed the salt
- CPU on which the salt was sealed

$\rightarrow$ Not sufficient to steal the password (protects against Evil Maid attacks)

Conclusion

Concept for isolating kernel components within Linux

- SGX does not work in ring zero
  → Move functionality to user space first
- Not generally applicable to kernel components
- Exemplary implementation for Linux FDE

SGX-kernel FDE

- Protects against cold-boot and DMA attacks
- Stronger than existing implementations such as TRESOR and ARMORED due to sealed salt
- Performance has to be improved

Thank you for your attention!

Further Information:

https://www1.cs.fau.de/sgx-kernel