Hacking in Darkness: Return-oriented Programming against Secure Enclaves

Dominik Pham

Technical University of Munich

December 2, 2018

Structure

1. Technical Background
   - Intel SGX
   - The ROP Attack
2. Dark-ROP Attack Design
   - Finding a vulnerability
   - Finding useful gadgets
3. The SGX Malware
   - Extracting hidden binary from enclave
   - Hijacking remote attestation as MitM
4. Mitigations

Intel SGX

Intel SGX - Security Features

- Memory encryption/isolation
- Program integrity through attestation
- Data sealing
- Deploying encrypted binary to enclave memory

Intel SGX - Instruction Specifications

User Instruction	Description
ENCLU[EENTER]	enter an enclave
ENCLU[EEXIT]	exit an enclave
ENCLU[EGETKEY]	create a cryptographic key
ENCLU[EREPORT]	create a cryptographic report
ENCLU[ERESUME]	re-enter an enclave

Table: The ENCLU instruction (index has to be stored in register *rax*).

Return Oriented Programming

- find function in with exploitable (buffer overflow) vulnerability
- exploit vulnerability to overwrite return address
  $\rightarrow$ attacker can execute any existing code (gadget)
  $\rightarrow$ attacker can chain gadgets to a ROP chain

ROP Attack against SGX

Problems:

- determine location of vulnerability in encrypted enclave is difficult
- determine location of gadgets in encrypted enclave is difficult

Dark-ROP Attack Design

Solution: Dark-ROP, a modified version of the ROP attack, which solves the mentioned problems

- Finding a buffer overflow vulnerability
- Finding gadgets to reuse

in an encrypted enclave binary

Dark-ROP - Finding a vulnerability

- enclave program has fixed number of entry points (usually functions)
- enumerate those functions and executes them with fuzzing arguments
- on memory corruption the fall-back routine Asynchronous Enclave Exit (AEX) is triggered
  → function is candidate for vulnerability
- AEX handler stores source address of page fault in register *cr2*

Dark-ROP - Finding gadgets

Requirements for enclave code:

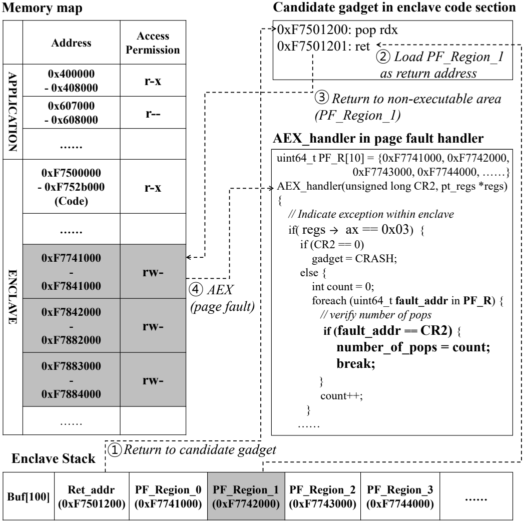
- must contain the ENCLU instruction
- must contain ROP gadgets with at least one *pop* instruction
- must contain function similar to *memcpy*

Dark-ROP - Finding *pop* gadgets

Page Fault oracle:

- probe through entire executeable address space of enclave memory
- after address to probe several non-executeable addresses (PF_Region_X)
- if address to probe is gadget with y *pops*, PF_Region_y is next return address
  $\rightarrow$ will trigger AEX with address of PF_Region_y in *cr2* register

Page Fault oracle:

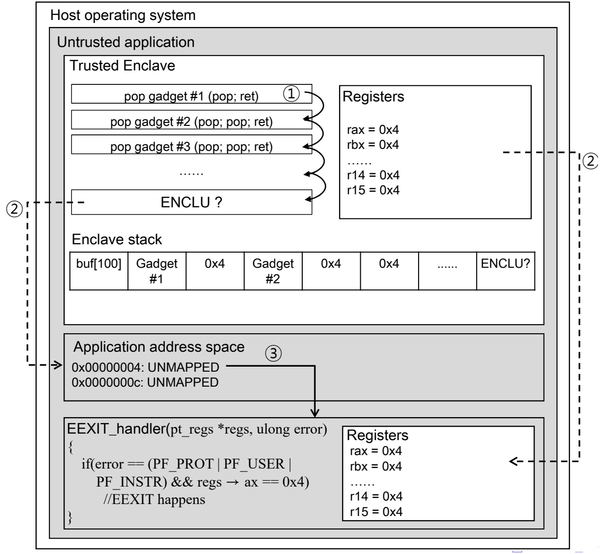


Dark-ROP - Finding *pop* gadgets

Identifying gadgets and registers oracle:

- find ENCLU instruction to call its EEXIT function
  $\leftrightarrow$ exiting enclave with this function will not clear registers
- chain *pop* gadgets with value 0x4 as every argument; address to probe at the end
- EEXIT function has an address as parameter stored in *rbx*
  $\rightarrow$ invoked if *rax* is 0x4 and address to probe is ENCLU
  $\rightarrow$ exception thrown if value in *rbx* is 0x4
- repeating with distinguishable values allows us to identify the *popped* registers

Identifying gadgets and registers oracle:



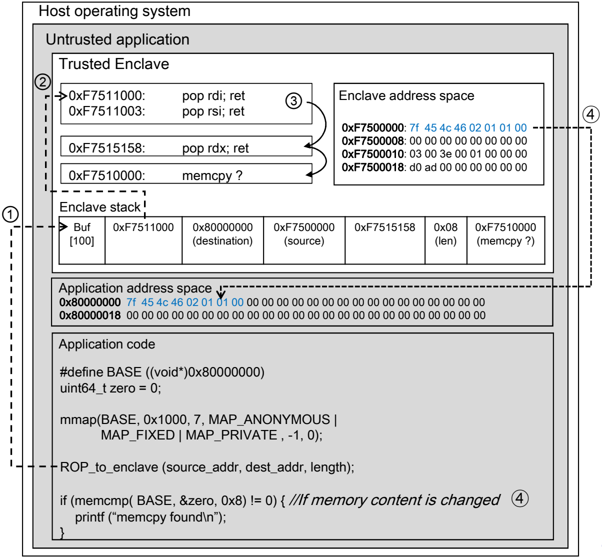
Dark-ROP - Finding *memcpy* gadgets

Read/Write gadget oracle:

- define source address in enclave address space *src* and a length *len*
- define destination address *dst* in untrusted memory space
  $\rightarrow$ set *dst* and next *len* bytes to zero
- chain *pop* gadgets to put *dst*, *src* and *len* in registers *rdi*, *rsi* and *rdx* with address to probe at the end
- if address to probe is *memcpy*, *dst* and next *len* bytes are non-zero

Dark-ROP - Finding *memcpy* gadgets

Read/Write gadget oracle:



Dark-ROP - Recap

We are now able to

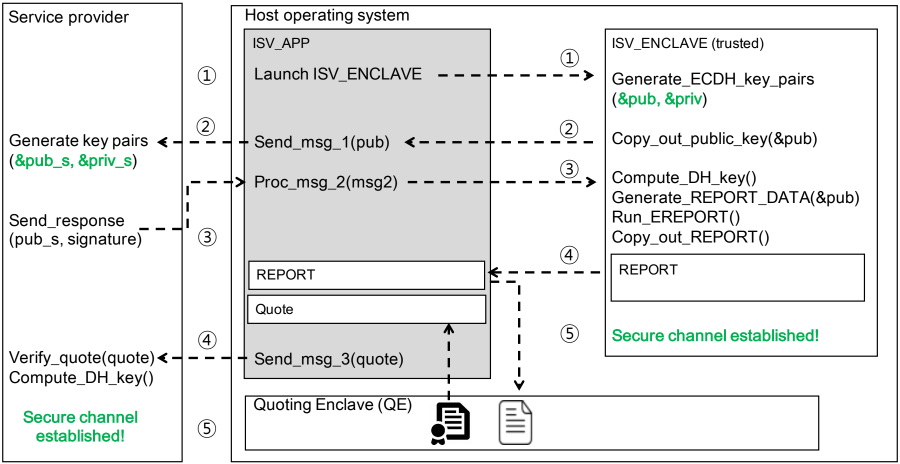
- call any leaf function through ENCLU
- set register values which are used as parameters in leaf functions
- copy data between the untrusted and trusted address space

The SGX Malware

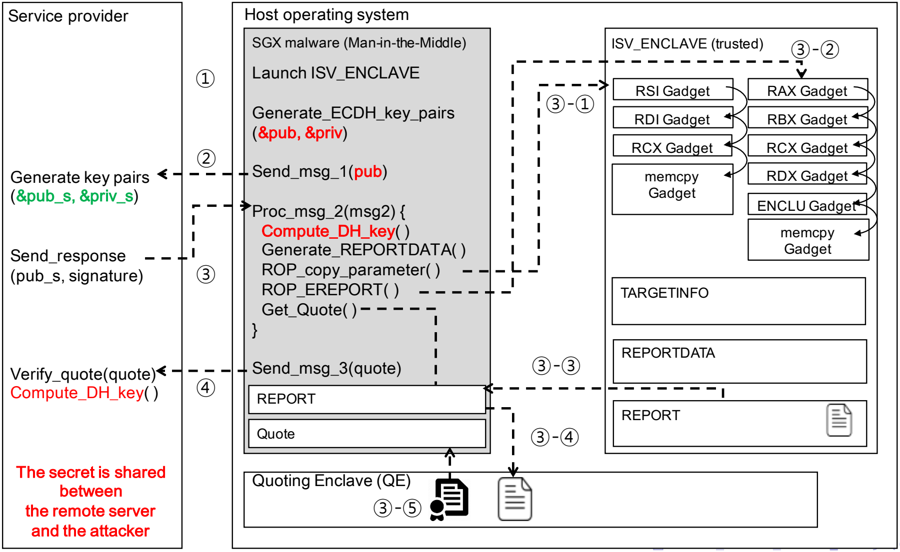
The SGX Malware - Extracting hidden binary from enclave

- utilizing *memcpy* gadget with
  $\rightarrow$ *src* as start of enclave's binary
  $\rightarrow$ *dst* as address in untrusted memory space
  $\rightarrow$ *len* as size of enclave's entire mapped space
- allows malware to mimic real enclave program
  $\leftrightarrow$ attacker can alter code for own purpose

Remote Attestation in SGX



The SGX Malware - Hijacking remote attestation as MitM



Mitigations

- Gadget elimination
  $\rightarrow$ modify enclave code to prevent non-intended *ret* instructions
  $\rightarrow$ for non-removeable gadgets: register validation after ENCLU
- Control flow integrity
  $\rightarrow$ should not use general registers for pointer

Thank you for your attention!