Controlled-Channel Attacks:
Deterministic Side Channels for Untrusted Operating Systems

Yuanzhong Xu, Weidong Cui, Marcus Peinado

Goal

- Protect the data of applications running on remote hardware

New tech

- Trusted Platform Modules – Limited use cases
  - Secure counters and signatures (trInc)
  - Undeniable access to data (Pasture)
  - State continuity (Memoir)

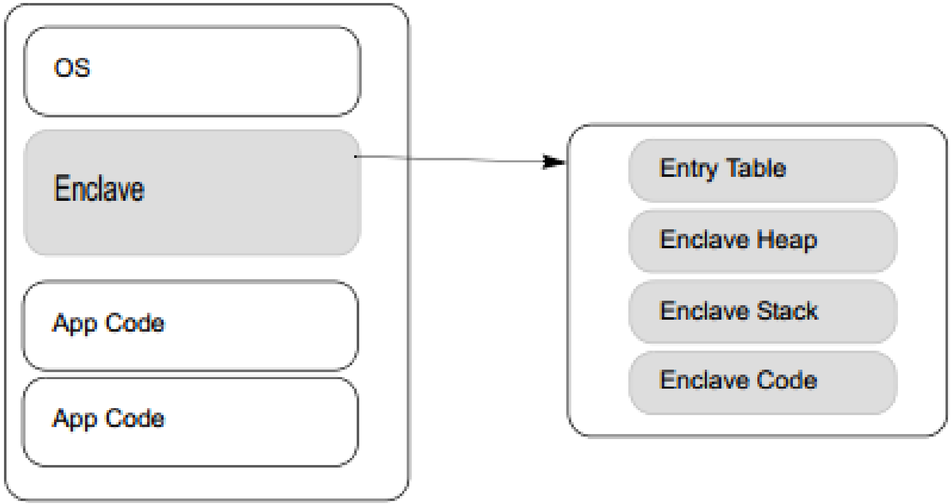
- Problems:
  - Need special hardware (costly)
  - Lack of flexibility – Need to redevelop applications (costly)

Intel Software Guard Extensions

- SGX
- New trusted hardware from Intel
- Tamper proof processor with burned in cryptographic keys for signing and encryption
- Protects user applications from adversarial operating systems
    - Cloud applications
    - Protecting users from themselves (Internet banking on compromised devices)

SGX Enclaves

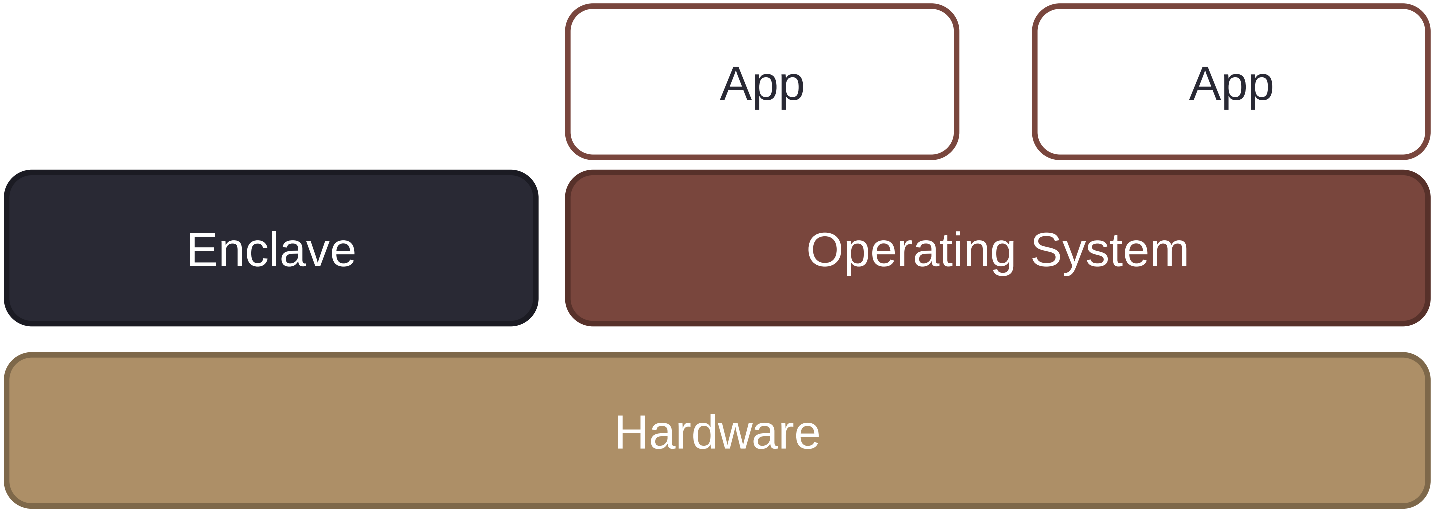
- Enclave = Secure execution context
  - Run a piece of code in encrypted memory



Source: Software Guard Extensions Reference Manual

SGX Enclaves

- Enclaves run directly on top of hardware



App

App

Enclave

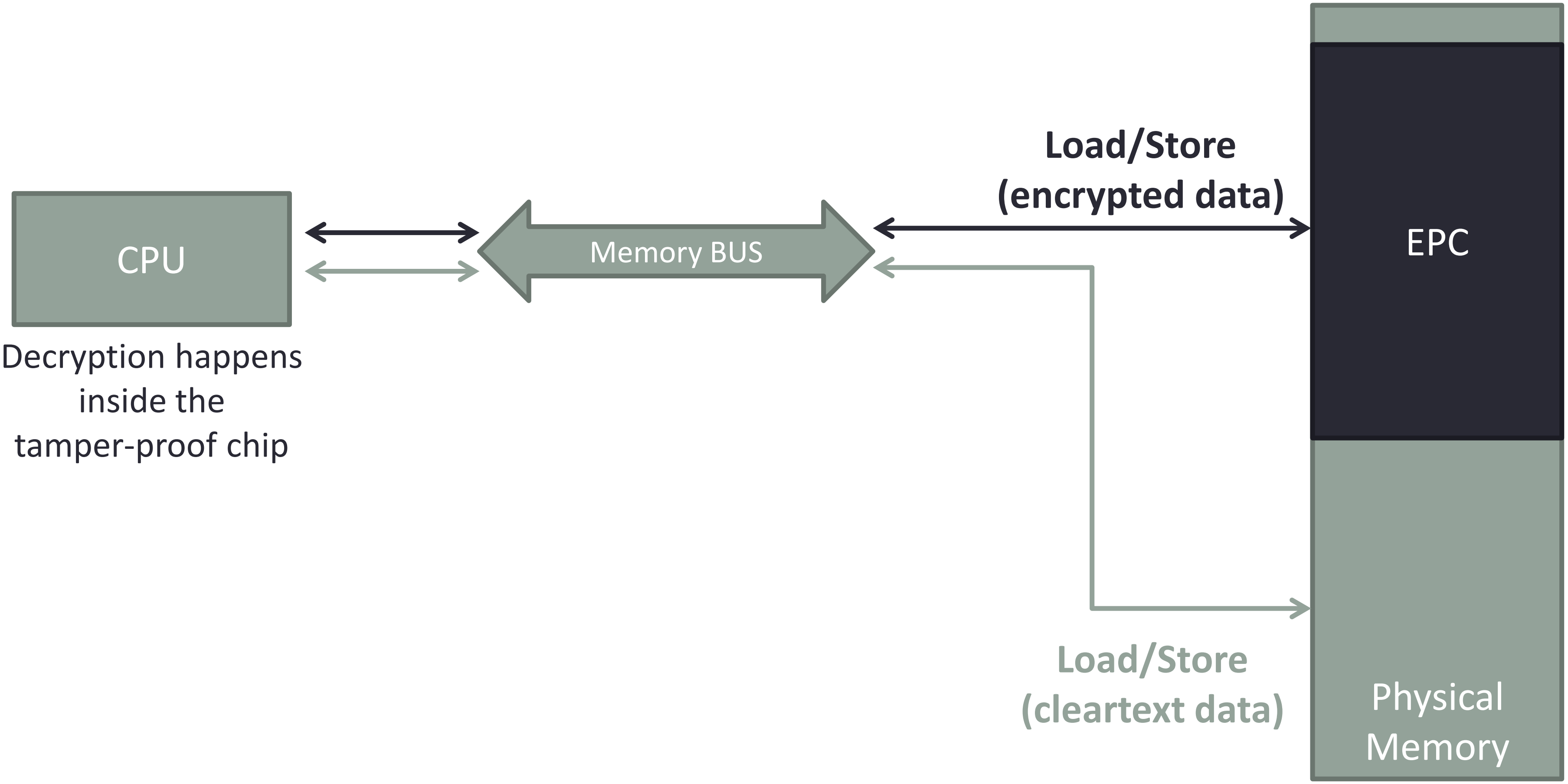
Operating System

Hardware

- Applications build the Enclave address space(ECREATE, EINIT)
  - The processor performs cryptographical checks to verify that the created enclave matches the developer’s intention
- Control is eventually passed to the enclave (EENTRY)
- The Enclave then executes on its own

SGX Enclaves

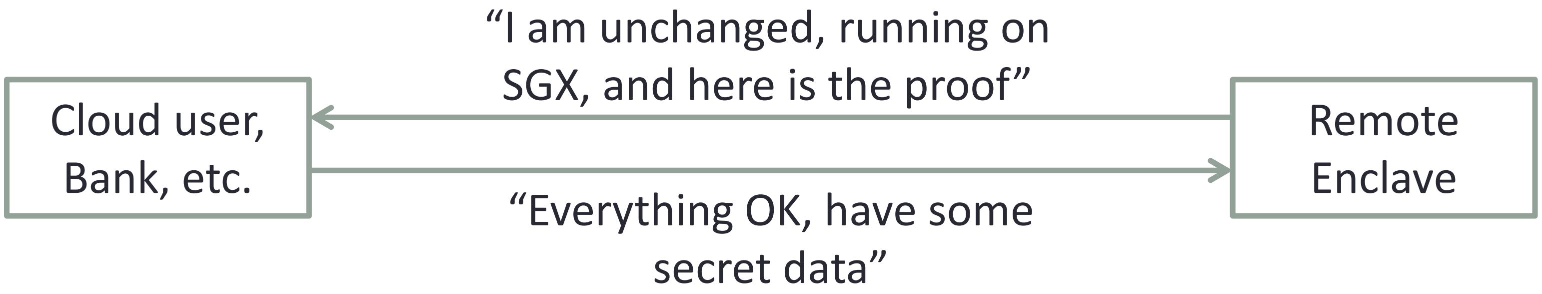
- Why are enclaves "secure"?
  - Enclave memory is stored within the Enclave Page Cache



Decryption happens inside the tamper-proof chip

SGX

- As an enclave, you can ask the processor to sign your execution context
  - Enclaves first prove themselves to third parties
- You pass secure data to the enclave only after it has "proven itself"
  - So you only pass data to (hopefully) secure code



Cloud user, Bank, etc.

"I am unchanged, running on SGX, and here is the proof"

"Everything OK, have some secret data"

Remote Enclave

Haven

- So we can run programs directly on top of secure hardware

- Next step?
  - Run entire operating systems on top of secure hardware
  - Final goal: Run generic, legacy applications, securely

- Major issue: operating system is needed to manage input and output

- Compromise: Run a user mode variant of an entire operating system
- Haven – The Windows API in a single process

Haven

- Run any legacy application on top of a library version of the Windows API

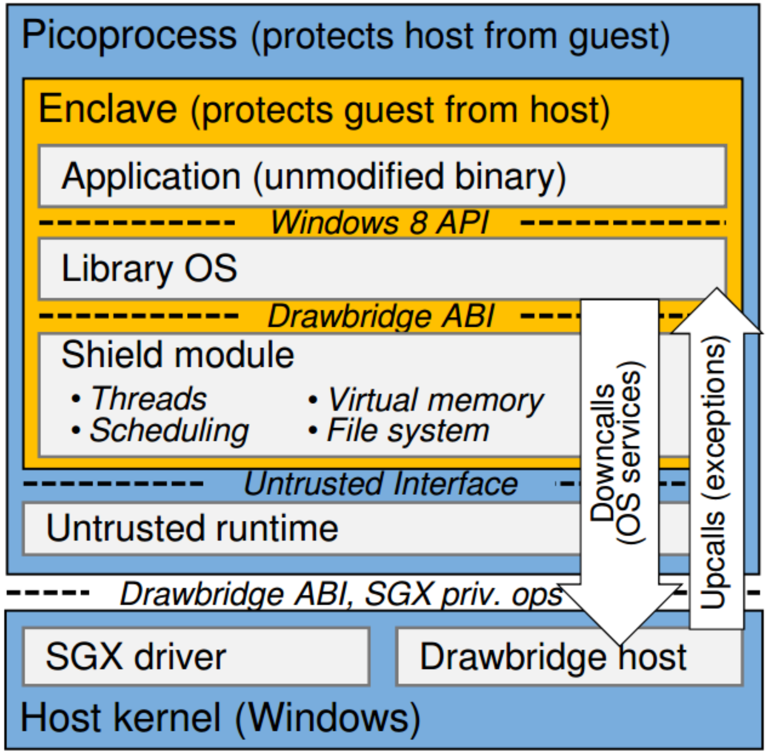


Image source: Baumann, A., Peinado, M. and Hunt, G., 2014, October. Shielding applications from an untrusted cloud with haven. In USENIX Symposium on Operating Systems Design and Implementation (OSDI).

Job done?

- We can now run secure applications with encrypted data on remote hardware

- As long as the keys within the SGX processor remain safe, the data remains encrypted
- SGX processor is also (according to Intel) resilient against side-channel attacks:
  - No power analysis
  - No timing analysis
  - No interesting radio waves

- Turns out it’s not…

Intermission: Virtual Memory

Virtual Memory

- Each process has access to its own virtual address space
- Separate physical memory from application memory

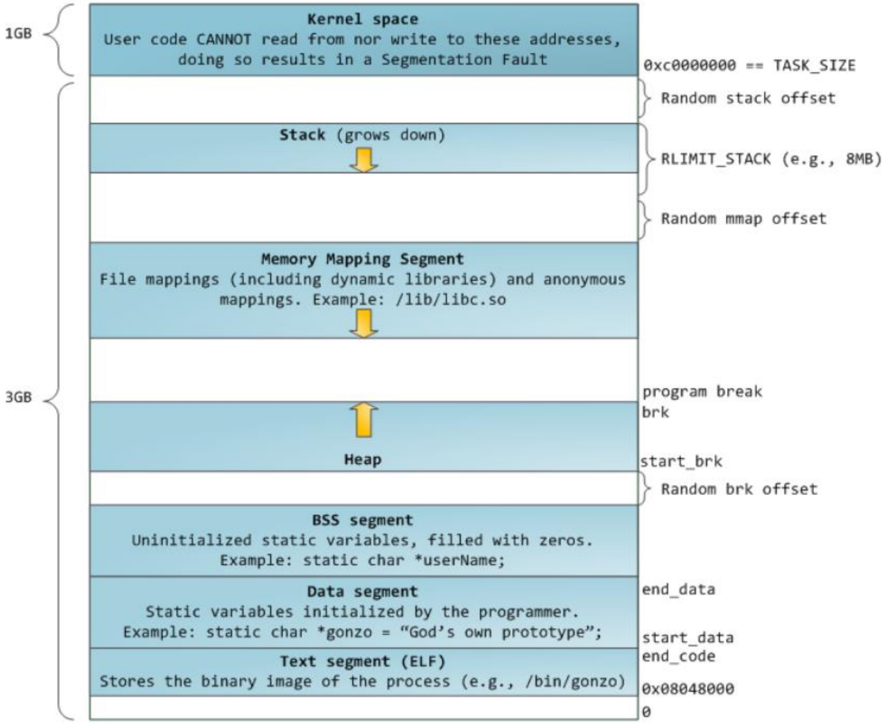
- Advantages:
  - Shared memory
  - Copy-on-write
  - Map files in memory

- Demand paging

Translated from: Operating Systems lecture slides, Faculty of Automatic Control and Computer Science, University POLITEHNICA of Bucharest

Virtual Memory – Pages

- No external fragmentation
- Smallest unit of allocation from the OS point of view
- Virtual pages (pages)
- Physical pages (frames)
- Map memory: associate frames to pages
- Memory Management Unit (MMU) handles translation



Translated from: Operating Systems lecture slides, Faculty of Automatic Control and Computer Science, University POLITEHNICA of Bucharest
Image from: http://duartes.org/gustavo/blog/post/anatomy-of-a-program-in-memory/

Virtual Memory – Page Tables

- Managed by the **operating system**
- Used by the **MMU** to perform translation
- Contain Page Table Entries that:
  - Point virtual memory addresses to physical memory addresses
  - Contain access rights (read, write, execute)

- If the MMU finds a problem → **page fault**

Translated from: Operating Systems lecture slides, Faculty of Automatic Control and Computer Science, University POLITEHNICA of Bucharest

Virtual Memory – Page Fault

- Access to a page that is
    - Unmapped
    - Invalid
    - Wrong access rights

- Exception is generated → Run page fault handler
- Page fault handler = Operating system

- **Red flag**: operating system was untrusted!
- **However**: operating system is managing pages with encrypted data, it can only perform denial of service attacks (?)

Translated from: Operating Systems lecture slides, Faculty of Automatic Control and Computer Science, University POLITEHNICA of Bucharest

Virtual Memory – Page Fault

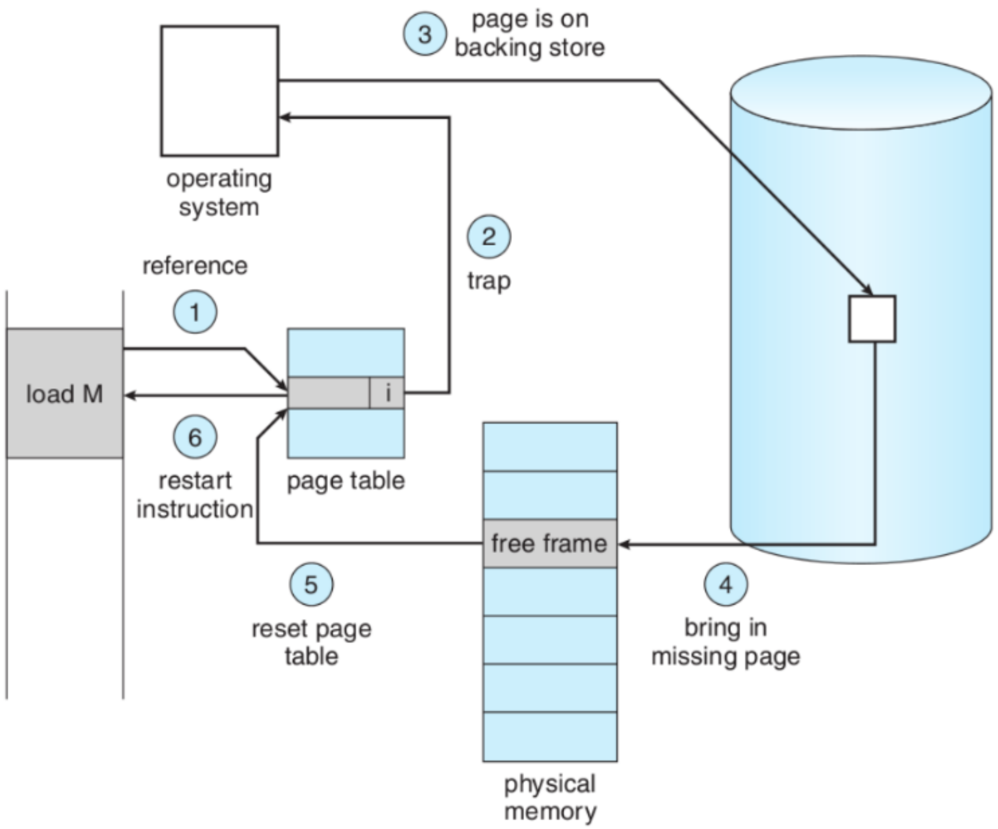


Image source: OSCE, Chapter 8, pg. 325, Figure 8.6

Controlled-Channel Attacks

Overview

- Problem: Page tables on SGX processors are still handled by the untrusted operating system
- In normal operation, the running enclave will cause a "small" amount of page faults
  - The MMU then reveals the required page to the operating system, so that the page fault handler can retrieve it
- The operating system can obtain valuable information from here
- Important assumption: **Application code is public**

Why it works

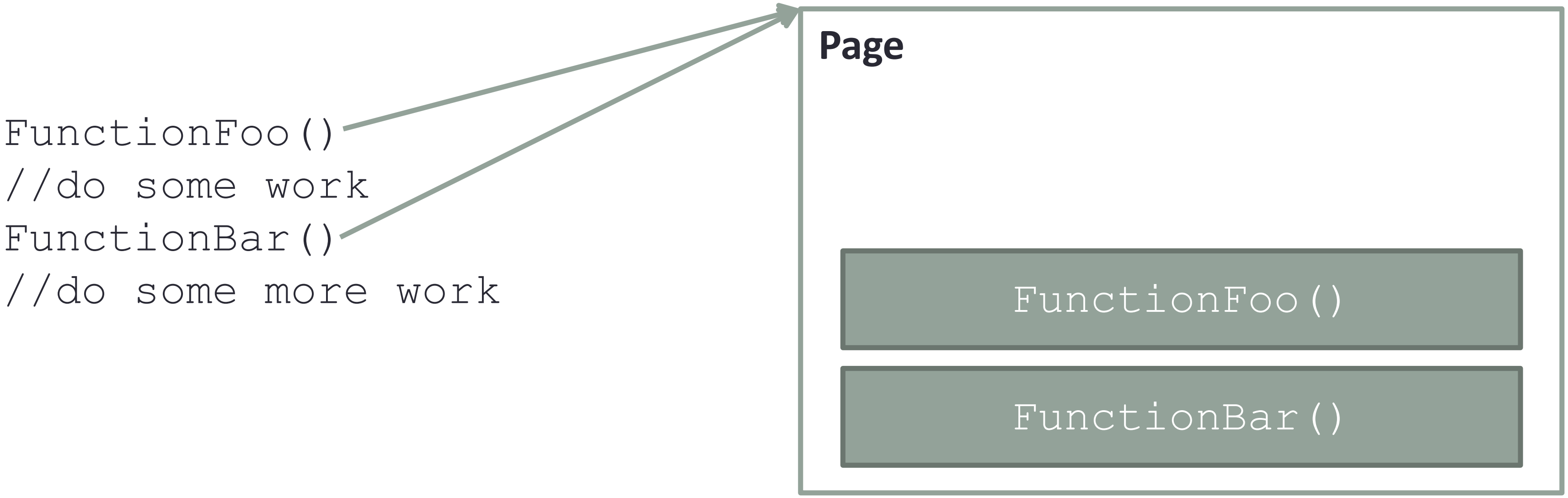
```
Char* WelcomeMessage( GENDER s ) {
        char *mesg;
        //GENDER is an enum of MALE and FEMALE
        if ( s == MALE ) {
                mesg = WelcomeMessageForMale();
        } else {
                mesg = WelcomeMessageForFemale();
        }
        return mesg;
}


Void CountLogin( GENDER s ) {
        if ( s == MALE ) {
                gMaleCount ++;
        } else {
                gFemaleCount ++;
        }
}
```

Page fault for address of **WelcomeMessageForMale** reveals **s** is MALE

Why it’s not trivial

- SGX tries to hide true page fault addresses by rounding them off to page boundaries
  - Page faults caused by accesses to multiple addresses (different functions, variables, etc.) look the same
  - Attack is still possible, only harder



```
FunctionFoo()
//do some work
FunctionBar()
//do some more work
```

Page

FunctionFoo()

FunctionBar()

Sources of information

- Control transfers
  - Different function are called, depending on the value of some variable

- Data accesses
  - Different variables are accessed, depending on the value of some other variable

  - Data accesses are monitored by looking at nearby code page faults
    - the instructions that use the data
  - Even dynamically allocated data accesses can be identified
    - the instruction patterns around them are the same

Overview

Before

- **Step 1**: Build an collection of page fault sequences to use as reference

During

- **Step 2**: After enclave start, remove access from all process pages
- **Step 3**: Record the pattern of page fault addresses

After

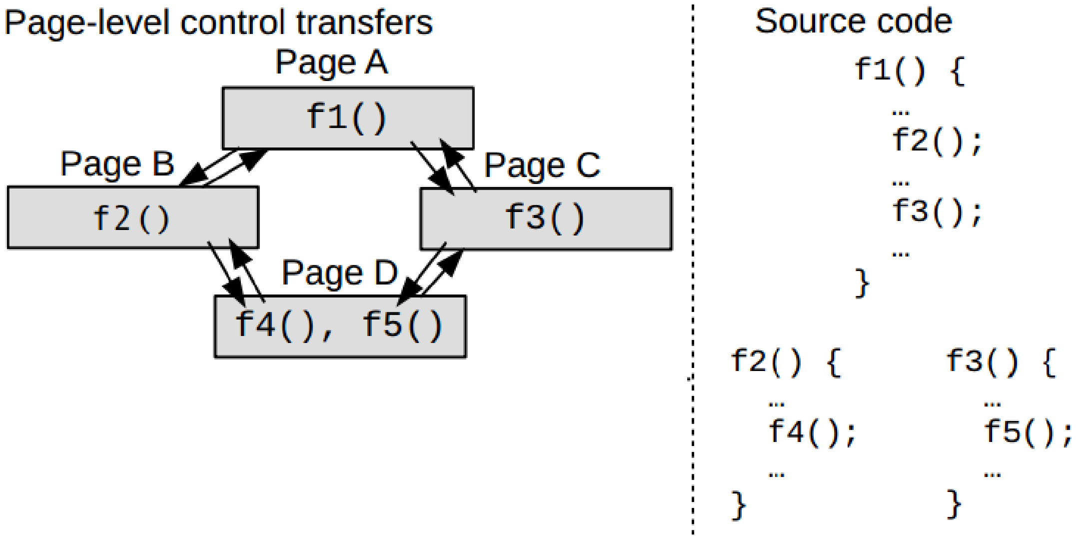
- **Step 4**: From the pattern, deduce information about data within the program

- Three attacks on common libraries:
  - Libjpeg
  - Hunspell
  - FreeType

Step 1

- Build an collection of page fault sequences to use as reference
- Record page faults of code pages
- Collect a set of page-fault traces $\left\{P_i = \left\{p_i^j\right\}\right\}$
- For each trace P_i, generate a new trace $Q_i = \left\{q_i^j\right\}$ that contains page base addresses
  - Q_i will be similar to the page faults provided by SGX
- For each s, t s.t. $p_s^t = f$, search for the minimum $k \geq 1$ such that for any sequence $\left(q_i^{j-k+1}, q_i^{j-k+2}, \ldots, q_i^j\right)$ that matches with the sequence $(q_s^{t-k+1}, q_s^{t-k+2}, \ldots, q_s^t)$, p_i^j equals to f
- Finally, use the set of unique sequences $\left\{\left(q_s^{t-k+1}, \ldots, q_s^t\right)\right\}$ to find the control transfer

Example – Control Transfers



```
f1() {
  …
  f2();
  …
  f3();
  …
}

f2() {        f3() {
  …             …
  f4();         f5();
  …             …
}             }
```

P_1	f1	f2	f4	f2	f1	f3	f5	f3	f1
Q_1	A	B	D	B	A	C	D	C	A

Image source: Xu, Y., Cui, W. and Peinado, M., 2015. Controlled-Channel Attacks: Deterministic Side Channels for Untrusted Operating Systems.

Example – Control Transfers

P_1	f1	f2	**f4**	f2	f1	f3	f5	f3	f1
Q_1	A	B	**D**	B	A	C	D	C	A

- Suppose we are searching for a jump to **`f4`**
  - At position 3
  - For position 3, search for the smallest $K \geq 1$ such that all K-length sequences in Q_1 matching the K-length sequence ending at position 3 in Q_1 end on **`f4`**
  - $K = 1$ doesn't work; why?

Same sequence (D) ends on f5

P_1	f1	f2	**f4**	f2	f1	f3	**f5**	f3	f1
Q_1	A	B	**D**	B	A	C	**D**	C	A

This sequence (B, D) is only found here

  - $K = 2$ is ok; why?

P_1	f1	**f2**	**f4**	f2	f1	f3	f5	f3	f1
Q_1	A	**B**	**D**	B	A	C	D	C	A

Steps 2 & 3

- After enclave start, remove access from all process pages
  - Access will cause a page fault

- Tracking all pages is too expensive → Track a subset of pages
  - Details on the algorithm are in the paper

- Upon receiving a fault, the handler:
  - Logs the requested page
  - Enables access to the page
  - Removes access to the previous page

Step 4

- Analyze the logs and find the desired control transfers
- This can be performed offline, after the online portion of the attack
- We search for the control transfer (i.e. calls to a certain function), and can finally extract data

```
if (x != 0) {
      f1();
} else {
      f2();
}
```

If found control transfer for `f1`, then `x != 0`

If found control transfer for `f2`, then `x != 0`

- Attacks on real libraries are feasible, but require some thought on how to recover relevant information

What about ASLR?

- Address Space Layout Randomization
  - Randomly arrange the address space of the process
- Could be implemented in the shielding system (e.g. Haven)
- Possible solution?

What about ASLR?

- Attack still possible:
  - Look at the first few code page faults on application start in an offline setting
  - In the live setting, try to match the new page fault addresses to the original ones, based on the corresponding sequence
  - Keep building the mapping between randomized pages and the original ones; eventually the heap, stack and libraries will be located

Results

- Demo applications using the targeted libraries
- FreeType font library:
  - The demo application rendered a book (taken as input in ASCII format) onto a bitmap
  - Result: The attack managed to extract the exact ASCII input
- Hunspell spellchecker:
  - The demo application spellchecked the same book
  - Result: The attack recovered approximately 75% of the words in the input, without punctuation
- Libjpeg library:
  - The demo application loaded jpeg images and saved them as bitmaps
  - Result: Recovered features of the input images

Results

- Frame of reference for colors could not be obtained, but contours could be observed

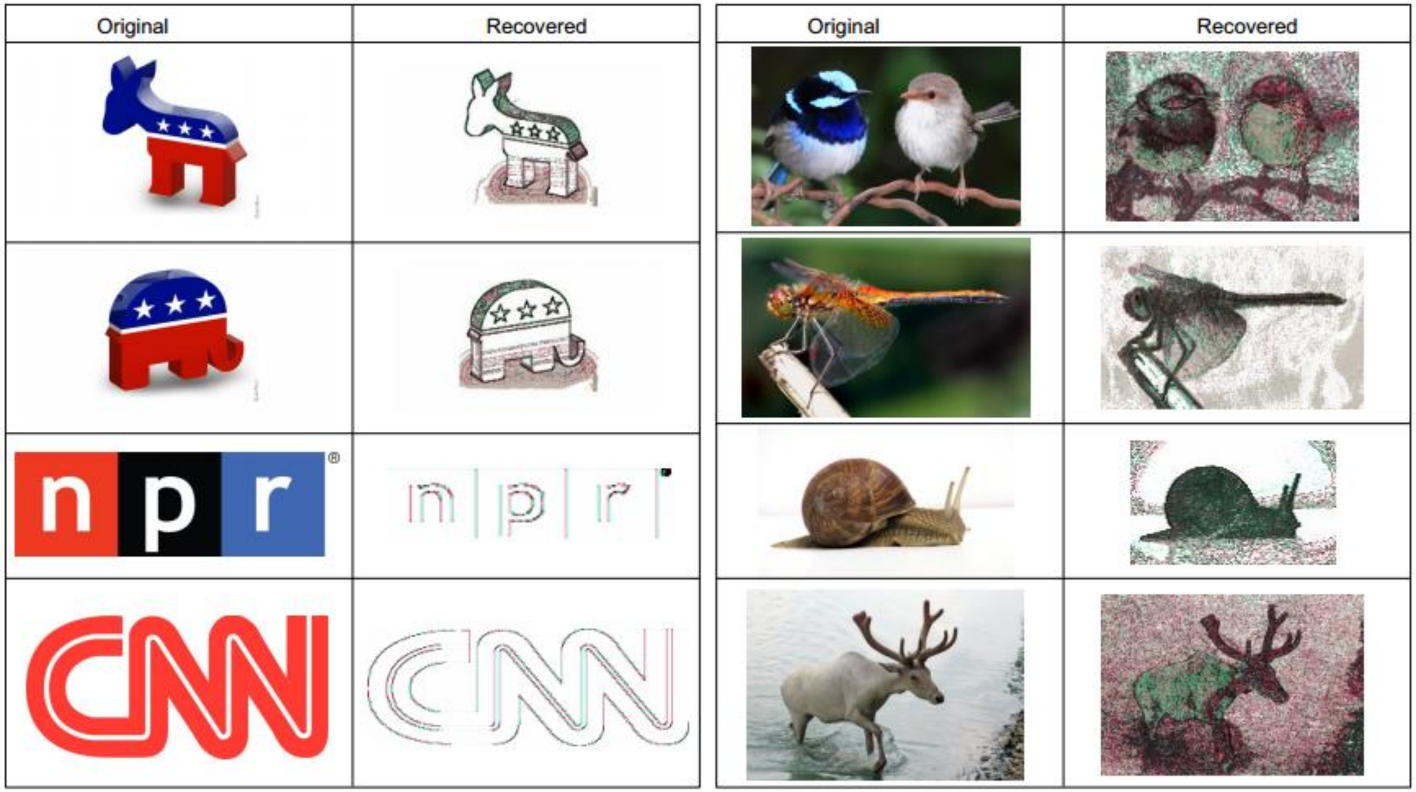


Image source: Xu, Y., Cui, W. and Peinado, M., 2015. Controlled-Channel Attacks: Deterministic Side Channels for Untrusted Operating Systems.

Conclusion

- Take home message: encryption on its own is not enough to protect executable code
  - Information can leak from many places