Securing Runtime Memory via MMU manipulation

Marinos Tsantekidis, Vassilis Prevelakis

Presenter: Marinos Tsantekidis

Institute of Computer Science - FORTH
Heraklion, Greece
E-mail: tsantekid@ics.forth.gr

TU Braunschweig
Germany
E-mail: tsantekidis@ida.ing.tu-bs.de

Short bio

I am Marinos Tsantekidis, a Ph.D. candidate at the Embedded Computer Security Workgroup of the Institute of Computer and Network Engineering at TU Braunschweig - Germany, under the supervision of Prof. Vassilis Prevelakis. I also hold a research assistant position at the Institute of Computer Science in FORTH - Greece. I received my Bachelor of Science degree in Computer Science from the Technological Educational Institute of Thessaloniki, Greece in 2011 and my Master of Science degree in Digital Systems Security from the University of Piraeus, Greece in 2015. My research focuses on security at the operating system level.

Current EU H2020 Projects:

CONCORDIA – Cyber security cOmpeteNCe fOr Research anD InnovAtion

https://www.concordia-h2020.eu

SENTINEL

https://sentinel-project.eu/

ROXANNE

https://www.roxanne-euproject.org/

Outline

Motivation

Design

Implementation

Evaluation

Conclusion

Motivation (1/3)

- New challenges and vulnerabilities every day
- Increasing requirements for security considerations and provisions for user applications
- Attackers more competent and effective
- Attacks more elaborate
- Complete security of a software system unfeasible
  - Detection of vulnerabilities before 0day attacks
  - Actively exploited vulnerabilities
- Program behavior monitoring
- Between OS and a running application

Motivation (2/3)

Two techniques for analysis, based on our previous work

1) Wrappers inserted between the program and the library code

2) Kernel intercepting transfers from main program to library or from library to library

Intercept all library calls from both the user as well as the kernel side, analyze them and take some form of action (reporting, argument checking, policy enforcement, etc.) before allowing them to continue

Motivation (3/3)

Trusted Execution Environment (TEE) at the memory space of a user application

- Isolated execution environment, parallel to a standard OS
  - Protection of sensitive code and data

MMU manipulation to map protected private pages into the address space of a running program, accessible only by specific functions inside external libraries

- Minimize what can access sensitive data/code
- Specific actions, specific parts of the program, specific point in execution time

Outline

Motivation

Design

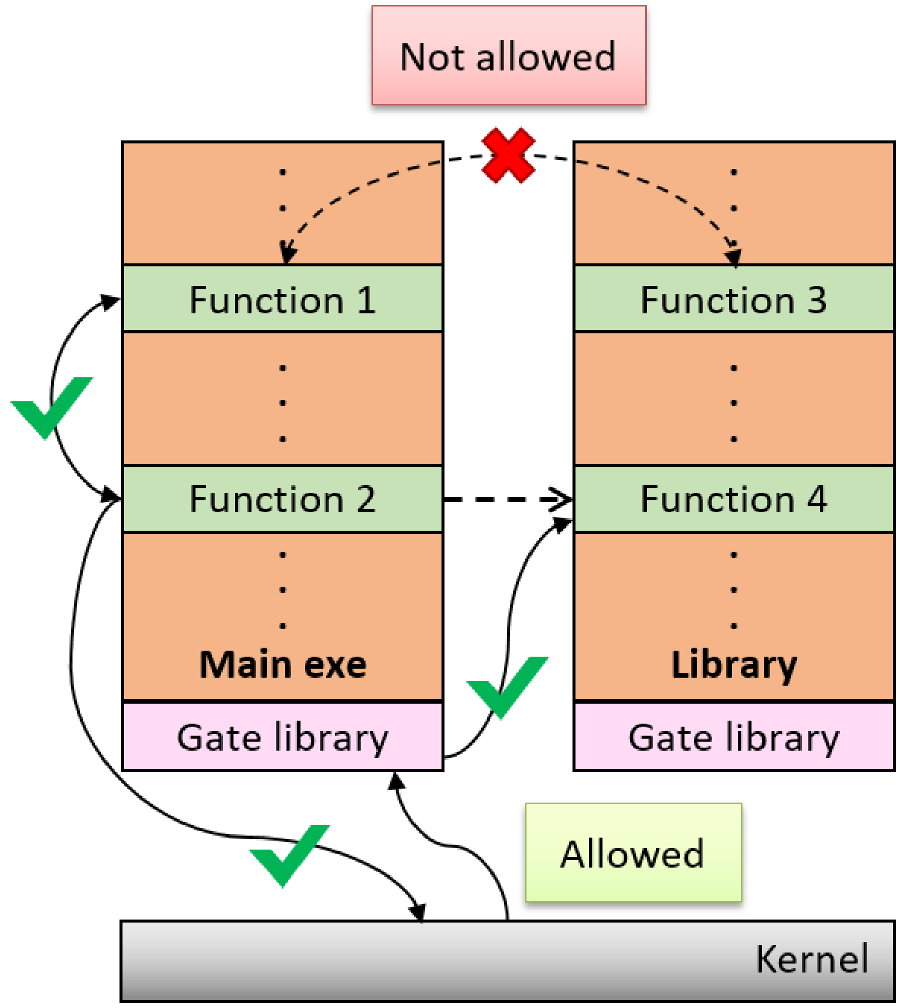
Implementation

Evaluation

Conclusion

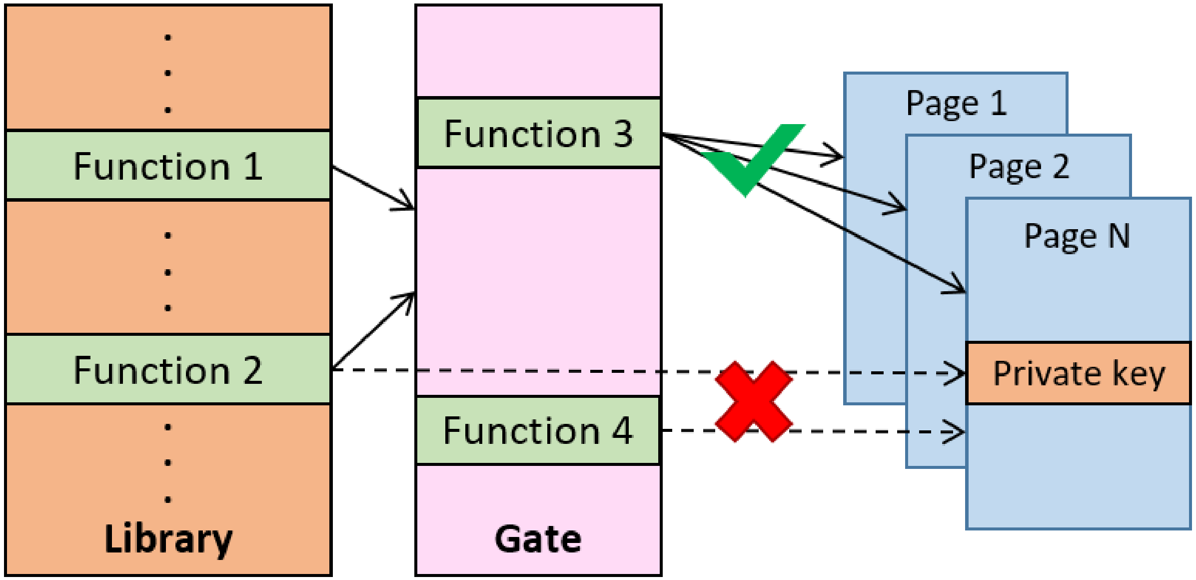
Design (1/2)

- Separate a process's executable areas (external libraries/main executable)
- Strict control over any attempt to invoke such an area
  - Redirect all calls through a *gate* library mapped by a custom Linux kernel (MMU)



Design (2/2)

- Map private secure memory pages for each area at run-time
- Accessible only to specific functions inside the *gate* library and only at specific intervals during execution



Outline

Motivation

Design

Implementation

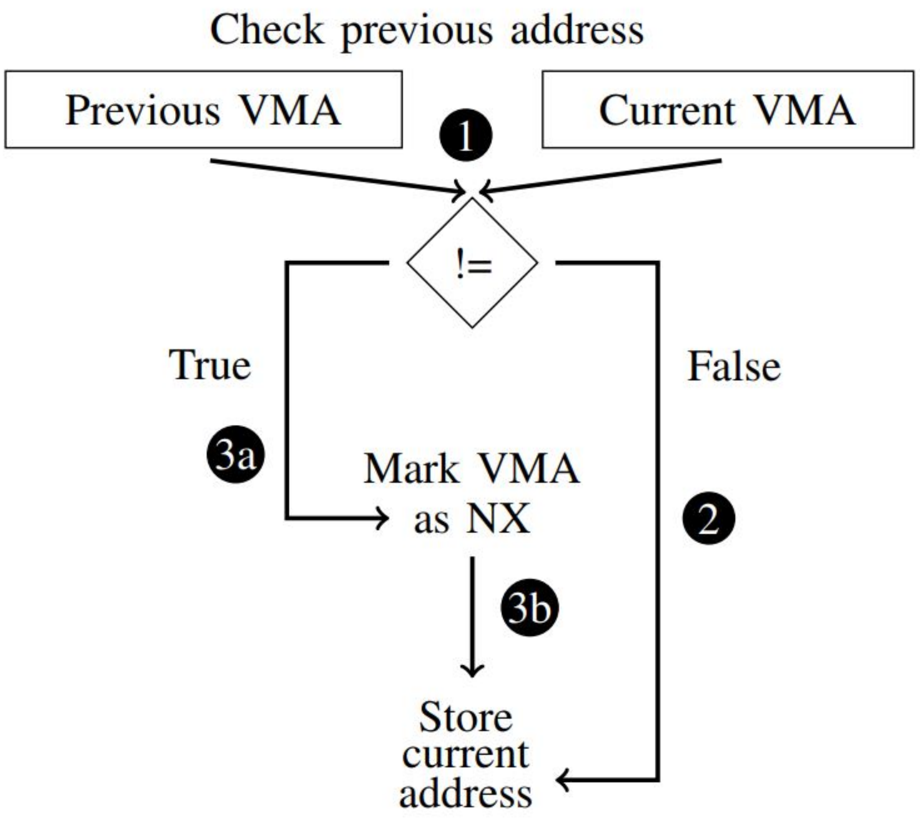
Evaluation

Conclusion

Implementation (1/3)

Compartmentalization

- Separate all executable Virtual Memory Areas (VMAs)
- Mark all the VMAs non-executable (NX)
- Map a custom *gate* library in the process's memory space, one for each identified VMA
- Intercept only calls between libraries and not internal ones



Implementation (2/3)

Private Memory Mapping

- Rewrite the Page Table whenever a library boundary is crossed
- Extend it by adding protected private memory pages for every library
- Mapped only when the CPU executes code within the associated library, otherwise unmapped
- Automatic transparent procedure
- No access to source code/binary required

Implementation (3/3)

Application Programming Interface (API)

- Analogous to the one used for shared memory

```
...
char *addr;
int fd;
fd = scrm_open(PAGE_SIZE, <FLAGS>);
addr = mmap(NULL, PAGE_SIZE,
            PROT_READ | PROT_WRITE,
            MAP_PRIVATE, fd, 0);
scrm_assoc(<caller>, fd, addr,
           addr + PAGE_SIZE);
...
scrm_unlink(fd);
...
```

Outline

Motivation

Design

Implementation

Evaluation

Conclusion

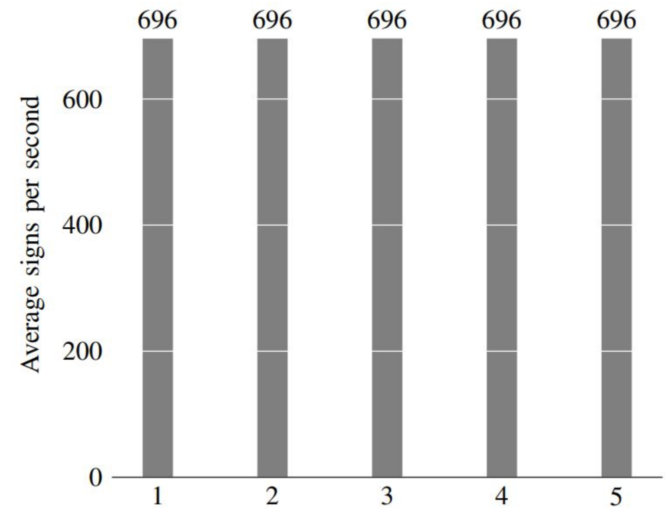
Evaluation (1/2)

Performance overhead

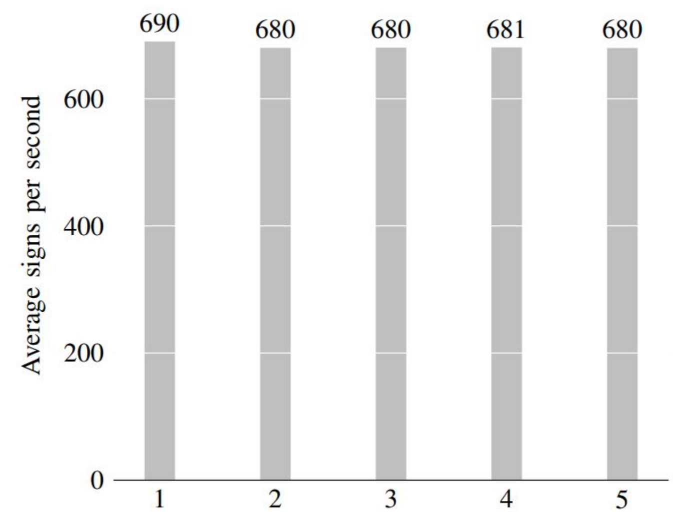
- Test-bed reported by PTS

PHORONIX-TEST-SUITE.COM	Phoronix Test Suite 9.0.1
AMD FX-8370 Eight-Core @ 4.00GHz (4 Cores / 8 Threads)	Processor
ASRock 970M Pro3 (P1.60 BIOS)	Motherboard
AMD RD9x0/RX980	Chipset
16384MB	Memory
256GB SAMSUNG MZ7TD256	Disk
Sapphire AMD Radeon HD 6450/7450/8450 / R5 230 OEM 1GB	Graphics
Realtek ALC892	Audio
DELL U2412M	Monitor
Realtek RTL8111/8168/8411	Network
Ubuntu 16.04	OS
4.16.7.CUSTOM (x86_64)	Kernel
Unity 7.4.5	Desktop
X Server 1.19.6	Display Server
modesetting 1.19.6	Display Driver
3.3 Mesa 18.0.5 (LLVM 6.0.0)	OpenGL
GCC 5.4.0 20160609	Compiler
ext4	File-System
1920x1200	Screen Resolution

- RSA 4096-bit signing
- 2% average decrease in performance



(a) Default kernel



(b) Custom kernel

Evaluation (2/2)

Memory Coverage Analysis

- Measure the degree of compartmentalization of a program's memory space
- Four well known, widely accepted applications

Library	Size (in bytes)	% of total
nginx (main)	528384	6.84%
libnss_files	45056	0.58%
libnss_nis	45056	0.58%
libnsl	90112	1.17%
libnss_compat	32768	0.42%
libdl	2093056	27.11%
libc	1835008	23.77%
libz	102400	1.33%
libcrypto	2207744	28.59%
libpcre	450560	5.84%
libcrypt	36864	0.48%
libpthread	98304	1.27%
ld	155648	2.02%
Total	7720960	100%

Application	Library	% of total
VMware Player	libvmwareui	21.53%
Sublime Text Editor	libgtk-3	20.63%
GNOME MPlayer	libicudata	34.74%

Outline

Motivation

Design

Implementation

Evaluation

Conclusion

Conclusion

- Monitoring framework based on two techniques from our previous work
  1) A library wrapper between a program and the original library code
  2) A kernel modification that intercepts all calls to libraries/executables
- TEE at the memory space of a user application
- Leverage MMU to map protected private pages into the address space of a running program
  - Limit what actions can be performed on the protected data, by what part of the program and at which point in execution time

Thank you

Questions?

https://mtsantekidis.gr