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
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:

https://www.concordia-h2020.eu
https://sentinel-project.eu/
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

```
Check previous address

Previous VMA | Current VMA

1. Check address
2. False
3a. Mark VMA as NX
3b. Store current address

True
```

previous address
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

```c
... 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);
...```

```c
```
Outline

Motivation
Design
Implementation
Evaluation
Conclusion
Evaluation (1/2)

Performance overhead

- Test-bed reported by PTS
- RSA 4096-bit signing
- 2% average decrease in performance
Evaluation (2/2)

Memory Coverage Analysis

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

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

<table>
<thead>
<tr>
<th>Application</th>
<th>Library</th>
<th>% of total</th>
</tr>
</thead>
<tbody>
<tr>
<td>VMware Player</td>
<td>libvmwareui</td>
<td>21.53%</td>
</tr>
<tr>
<td>Sublime Text Editor</td>
<td>libgtk-3</td>
<td>20.63%</td>
</tr>
<tr>
<td>GNOME MPlayer</td>
<td>libcudadata</td>
<td>34.74%</td>
</tr>
</tbody>
</table>
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