Persistent Data-Only Malware
Function Hooks Without Code

Sebastian Vogl, Jonas Pfoh, Thomas Kittel, and Claudia Eckert
{vogls,pfoh,kittel,eckert}@sec.in.tum.de

Technische Universität München
Chair for IT-Security
Munich, Germany

26.02.2014
1. Motivation & Background

2. Persistent data-only malware

3. Proof of Concept

4. Conclusion
Motivation & Background

- Problem

Protection Mechanisms
Motivation & Background

Problem

Protection Mechanisms

- $W \oplus X$
Motivation & Background

- Problem

Protection Mechanisms

- $W \oplus X$
- Signed Driver Loading
Protection Mechanisms

- $W \oplus X$
- Signed Driver Loading
- Secure Boot
Motivation & Background

Problem

Protection Mechanisms

- $\mathcal{W} \oplus \mathcal{X}$
- Signed Driver Loading
- Secure Boot
- Code Integrity Checking
Motivation & Background

Problem

Protection Mechanisms

- $W \oplus X$
- Signed Driver Loading
- Secure Boot
- Code Integrity Checking

⇒ It is getting more and more difficult to introduce malicious **code** into the system.
Data-only Malware
Motivation & Background

Solution

Data-only Malware

Hund et. al [1]: Return-oriented Rootkits (2009)
Motivation & Background

- Return-oriented Programming (ROP)

```
Gadget @ 0x5000
Gadget @ 0x4000
0x8000: ADD EAX, EBX;
0x8003: RET;
0x7276: RET;
```

```
EAX 0x41
EBX 0x1
EIP 0x7276
```

```
Register

Stack
```

Gadgets

```
0x4000
0x5000
```

```
0x8000
0x8003
0x7276
```

```
... 0x5000
... 0x4000
... 0x8000
... 0x7276
```

ESP
Motivation & Background

- Return-oriented Programming (ROP)

```
0x41 0x1
0x8000
...
Gadget @ 0x5000
0x4000
0x8000: ADD EAX,EBX;
0x8003: RET;
0x7276: RET;
```

Gadget @ 0x5000
Gadget @ 0x4000

```
0x5000
0x4000
0x8000
...
```

ESP
Motivation & Background

- Return-oriented Programming (ROP)

## Register

- **EAX**: 0x42
- **EBX**: 0x1
- **EIP**: 0x8003

## Gadgets

- **Gadget @ 0x5000**
- **Gadget @ 0x4000**
- **0x8000: ADD EAX,EBX;**
- **0x8003: RET;**
- **0x7276: RET;**

## Stack

- **ESP**
- **0x5000**
- **0x4000**
- **0x8000**
- **...**
**Motivation & Background**

- Return-oriented Programming (ROP)

### Register

- **EAX**: 0x42
- **EBX**: 0x1
- **EIP**: 0x4000

### Gadgets

- **Gadget @ 0x5000**
- **Gadget @ 0x4000**
- 0x8000: ADD EAX,EBX;
  0x8003: RET;
- 0x7276: RET;

### Stack

- ESP

- **0x5000**
- **0x4000**
- **0x8000**
- ...
Data-only Malware

Hund et. al [1]: Return-oriented Rootkits (2009)
Motivation & Background

Solution

Data-only Malware

Hund et. al [1]: Return-oriented Rootkits (2009)

▶ “One-Shot-Attacks”
Hund et. al [1]: Return-oriented Rootkits (2009)

- “One-Shot-Attacks”
- **Triggered** by attacker
Motivation & Background

- Persistence

- Cannot **intercept events** within the system
Motivation & Background

- Persistence

- Cannot intercept events within the system
- Usually accomplished with hooks
Motivation & Background

- Persistence

- Cannot **intercept events** within the system
- Usually accomplished with **hooks**
- **Where** should these hooks point to?
Motivation & Background

- Persistence

- Cannot **intercept events** within the system
- Usually accomplished with **hooks**
- **Where** should these hooks point to?

**Persistence?**
Outline

1 Motivation & Background

2 Persistent data-only malware

3 Proof of Concept

4 Conclusion
Finding a suitable location for the persistent control structure
Persistent data-only malware

- Finding a suitable location for the persistent control structure

- Control structure must be *exclusively* owned by the malware
Control structure must be **exclusively** owned by the malware
⇒ Create **new** memory region (e.g. kmalloc)
Persistent data-only malware

- Challenges

1. Finding a suitable **location** for the **persistent** control structure
2. Protecting against **overwrites**
Persistent data-only malware

- Protecting against overwrites

Register

Gadgets

A+0x0: ...
A+0x3: CALL ECX;
A+0x6: ...

Stack

...  

A

B

C

...  

ESP

ECX  

X  

EIP  

A + 0x3
Persistent data-only malware

- Protecting against overwrites

Register

<table>
<thead>
<tr>
<th>ECX</th>
<th>EIP</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>A + 0x3</td>
</tr>
</tbody>
</table>

Gadgets

- A+0x0: ...
- A+0x3: CALL ECX;
- A+0x6: ...

Stack

- A
- B
- C
- ...

ESP

Vogl, Pfoh, Kittel, Eckert (TUM)
Persistent data-only malware

- Protecting against overwrites

Register

ECX

EIP

Gadgets

A+0x0: ...
A+0x3: CALL ECX;
A+0x6: ...

Stack

ISP

A+6

B

C

...
Two types of overwrites:
  - **Interrupt-induced** overwrites
Persistent data-only malware

- Protecting against overwrites

Two types of overwrites:
  - **Interrupt-induced** overwrites
    - *Disable* interrupts
Two types of overwrites:

- **Interrupt-induced** overwrites
  - *Disable* interrupts
  - No *external* function calls
Persistent data-only malware

- Protecting against overwrites

Two types of overwrites:

- **Interrupt-induced** overwrites
  - *Disable* interrupts
  - No *external* function calls

- **Self-induced** overwrites
Persistent data-only malware

- Protecting against overwrites

Two types of overwrites:
  - **Interrupt-induced** overwrites
    - *Disable* interrupts
    - No *external* function calls
  - **Self-induced** overwrites
    - *Carefully* design persistent chain
Persistent data-only malware

- Challenges

1. Finding a suitable location for the persistent control structure
2. Protecting against overwrites
3. Resuming the original control flow
Persistent data-only malware

- Resuming the original control flow

- Registers must be saved **before use**
Resuming the original control flow

- Registers must be saved **before use**
- Control flow must be **restored** after execution
Persistent data-only malware

- Challenges

1. Finding a suitable **location** for the **persistent** control structure
2. Protecting against **overwrites**
3. **Resuming** the original control flow
4. **Activating** the **persistent** control structure
Persistent data-only malware

- Activating the persistent control structure

- *Control structures* somewhere in memory
Persistent data-only malware

- Activating the persistent control structure

  - Control structures **somewhere** in memory
  - Only control the **Instruction Pointer** when a hook is triggered
Persistent data-only malware

- Activating the persistent control structure

- *Control structures* somewhere in memory
- Only control the *Instruction Pointer* when a hook is triggered
- Must not use *general purpose registers* (backup!) for the switch
Persistent data-only malware

- Activating the persistent control structure

- *Control structures* *somewhere* in memory
- Only control the **Instruction Pointer** when a hook is triggered
- Must not use **general purpose registers** (backup!) for the switch

Solution

sysenter
Persistent data-only malware

- Activating the persistent control structure
**Victim**  Ubuntu 64-bit Server (Kernel 3.8) with secure boot (UEFI)
**Victim**  Ubuntu 64-bit Server (Kernel 3.8) with secure boot (UEFI)

**Vulnerability**  CVE-2013-2094 Local Root Exploit
Data-only version
**Victim**  Ubuntu 64-bit Server (Kernel 3.8) with secure boot (UEFI)

**Vulnerability**  CVE-2013-2094 Local Root Exploit
Data-only version

**Hooks**  sys_read, sys_getdents
Proof of Concept

**Victim**  Ubuntu 64-bit Server (Kernel 3.8) with secure boot (UEFI)

**Vulnerability**  CVE-2013-2094 Local Root Exploit Data-only version

**Hooks**  sys_read, sys_getdents

**Functionality**  Key logging, process hiding, file hiding
Persistent data-only malware is possible
Conclusion

- **Persistent** data-only malware is possible
- Mainly **technical** challenges
Conclusion

- **Persistent** data-only malware is possible
- Mainly **technical** challenges
- Data-only malware is a **realistic** threat
Persistent data-only malware is possible

Mainly technical challenges

Data-only malware is a realistic threat

POC code available on our website
http://www.sec.in.tum.de/persistent-data-only-malware/
Ralf Hund, Thorsten Holz, and Felix C. Freiling.
Return-oriented rootkits: Bypassing kernel code integrity protection mechanisms.