PIkit: A New Kernel-Independent Processor-Interconnect Rootkit

August 10, 2016
Wonjun Song, Hyunwoo Choi, Junhong Kim, Eunsoo Kim, Yongdae Kim, John Kim
Korea Advanced Institute of Science and Technology
Rootkit Background

- **Rootkit**: A malicious software running on compromised machines without being detected.

- Typical root attack scenario

- Different types of rootkits by payloads and by vulnerabilities
# Rootkit Classification

<table>
<thead>
<tr>
<th>Vulnerabilities</th>
<th>Malicious Payloads</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Software</strong></td>
<td><strong>Kernel-Level</strong></td>
</tr>
<tr>
<td>T0rn (SANS ’00), Lrk5 (‘00), dica (’02), etc.</td>
<td>ROR (USESEC ’09), DKOM (BLACK HAT ’04), knark (‘99), etc.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Hardware</strong></th>
<th><strong>This work (PIkit)</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Cloaker (S&amp;P ’08), Shadow Walker (BLACK HAT ‘05)</td>
</tr>
</tbody>
</table>
High-level overview of PIkit

Kernel-Independent
(No code modification or injection)

Vulnerable hardware feature
(Processor-interconnect in x86 multi-socket server)

A very stealthy rootkit
(only simple read/write memory operations)

Kernel

Memory-Mapped Registers

0x..10 DRAM Address Mapping

Malicious Payload

User

Before/After PIkit installation
Analysis of Processor-Interconnect in Multi-Socket Servers
Multi-socket (NUMA) Server

Processor-Interconnect
- AMD HyperTransport (HT)
- Intel Quick-Path Interconnect (QPI)

* DELL PowerEdge 6950
Multi-socket Server Market Share

2-ways Intel multi-sockets (Fujitsu RX2540)
8-ways Intel multi-sockets (DELL PowerEdge C6100)
4-ways AMD multi-sockets (DELL PowerEdge R815)

*In datacenters and high-performance computing, over 80% of the x86 server are multi-socket servers (from IDC)*
Processor-Interconnect Overview

*MC: Memory Controller
Memory access in Processor-Interconnect

DRAM Address Mapping Table

<table>
<thead>
<tr>
<th>Base Address</th>
<th>Limit Address</th>
<th>Destination ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0x0000000000</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0x0420000000</td>
<td>North</td>
</tr>
<tr>
<td>2</td>
<td>0x0C20000000</td>
<td>Local</td>
</tr>
<tr>
<td>3</td>
<td>0x101F000000</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>RESERVED</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>RESERVED</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>RESERVED</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>RESERVED</td>
<td></td>
</tr>
</tbody>
</table>

If we are able to modify the DRAM mapping table, we can send the packet to wherever we want.
### DRAM Address Mapping Table

<table>
<thead>
<tr>
<th>Base Address</th>
<th>Limit Address</th>
<th>Destination ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000000000</td>
<td>0x041F000000</td>
<td>0</td>
</tr>
<tr>
<td>0x0420000000</td>
<td>0x081F000000</td>
<td>1</td>
</tr>
<tr>
<td>0x0820000000</td>
<td>0x0C1F000000</td>
<td>2</td>
</tr>
<tr>
<td>0x0C20000000</td>
<td>0x101F000000</td>
<td>3</td>
</tr>
<tr>
<td>RESERVED</td>
<td>RESERVED</td>
<td>RESERVED</td>
</tr>
<tr>
<td>RESERVED</td>
<td>RESERVED</td>
<td>RESERVED</td>
</tr>
<tr>
<td>RESERVED</td>
<td>RESERVED</td>
<td>RESERVED</td>
</tr>
<tr>
<td>RESERVED</td>
<td>RESERVED</td>
<td>RESERVED</td>
</tr>
</tbody>
</table>

- **Configurability**
- **Discrepancy**
- **Extra entries**
- **No integrity checking**
PIkit Design & Implementation
High-Level Overview of PIkit installation

SOFTWARE STACK

- USERLEVEL APPLICATION
- OPERATING SYSTEM
- ROOKIT MONITORING

SOFTWARE STACK

- APPLICATION LAYER
- SYSTEM SOFTWARE LAYER

Hardware

- Defining Attack Address Region
- Modifying DRAM Address Mapping Table

Main Memory

- NODE 0 memory: 0x0...000
- NODE 1 memory: NODE 1 memory
- NODE 2 memory: NODE 2 memory
- NODE 3 memory: NODE 3 memory
- NODE 0 memory: NODE 0 memory
- NODE 1 memory: NODE 1 memory
- NODE 2 memory: NODE 2 memory
- NODE 3 memory: NODE 3 memory

Cannot detect PIkit at software-level
Defining Attack Address Region

- Only the attacker should have access to a particular memory region.
  - To prevent any unknown system behaviors (system crash)

- The memory range of the attack address region needs to be equal to resolution of the memory mapping table.
  - ex) AMD Opteron 6128: 16 MB
  - Can take advantage of huge pages (malloc for 1 GB huge page)

- The process that received the memory allocation is continuously running.
Modifying DRAM Address Mapping Table

- Need to translate **VA** (virtual address) to **PA** (physical address)
  - Attack Address Region: VA, DRAM Address Mapping: PA
  - e.g. /proc/(pid)/pagemap

- **Memory-mapped register** (AMD: 8 set of DRAM Base/Limit Registers)
  - DRAM Base Address Register
  - DRAM Limit Address Register

With root permission, the registers can be modified by using system read/write commands (eg. setpci)
How to modify DRAM Address Mapping Table

<table>
<thead>
<tr>
<th>Base Address</th>
<th>Limit Address</th>
<th>Dest ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000000000</td>
<td>0x041F000000</td>
<td>0</td>
</tr>
<tr>
<td>0x0420000000</td>
<td>0x081F000000</td>
<td>1</td>
</tr>
<tr>
<td>0x0820000000</td>
<td>0x0C1F000000</td>
<td>2</td>
</tr>
<tr>
<td>0x0C20000000</td>
<td>0x101F000000</td>
<td>3</td>
</tr>
<tr>
<td>0x07C0000000</td>
<td>0x07BF000000</td>
<td>1</td>
</tr>
<tr>
<td>0x07C0000000</td>
<td>0x07C1000000</td>
<td>2</td>
</tr>
<tr>
<td>0x0420000000</td>
<td>0x07BF000000</td>
<td>1</td>
</tr>
<tr>
<td>0x07C2000000</td>
<td>0x081F000000</td>
<td>1</td>
</tr>
<tr>
<td>RESERVED</td>
<td>RESERVED</td>
<td>RESERVED</td>
</tr>
</tbody>
</table>
Example – AMD Opteron 6128 server

**DRAM Address Mapping Table**

<table>
<thead>
<tr>
<th>Base Addr</th>
<th>Limit Addr</th>
<th>Dest Node ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000</td>
<td>0x041F</td>
<td>0</td>
</tr>
<tr>
<td><strong>RESERVED</strong></td>
<td><strong>RESERVED</strong></td>
<td><strong>RESERVED</strong></td>
</tr>
<tr>
<td>0x0820</td>
<td>0x0C1F</td>
<td>2</td>
</tr>
<tr>
<td>0x0C20</td>
<td>0x101F</td>
<td>3</td>
</tr>
<tr>
<td>0x0420</td>
<td>0x07BF</td>
<td>1</td>
</tr>
<tr>
<td>0x07C0</td>
<td>0x07C1</td>
<td>2</td>
</tr>
<tr>
<td>0x07C2</td>
<td>0x081F</td>
<td>1</td>
</tr>
<tr>
<td>Reserved</td>
<td>Reserved</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

**Routing Table**

<table>
<thead>
<tr>
<th>Dest Node ID</th>
<th>Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>North</td>
</tr>
<tr>
<td>1</td>
<td>Local</td>
</tr>
<tr>
<td>2</td>
<td>East</td>
</tr>
<tr>
<td>3</td>
<td>South</td>
</tr>
</tbody>
</table>

**Attack Region:**

0x07C0 – 0x07C1
## Extending PIkit to Intel Architecture

<table>
<thead>
<tr>
<th></th>
<th>AMD Opteron 6128</th>
<th>Intel 7500 series</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory-mapped Registers</td>
<td>DRAM Base/Limit registers</td>
<td>Source Address Decoder (SAD), Target Address Decoder (TAD)</td>
</tr>
<tr>
<td>Lookup Location</td>
<td>Source node</td>
<td>Source node, Destination node</td>
</tr>
<tr>
<td>Granularity</td>
<td>16MB</td>
<td>64MB</td>
</tr>
<tr>
<td># of entries</td>
<td>8</td>
<td>20</td>
</tr>
</tbody>
</table>
Based on the offset value, a fine-grained attack possible.
Malicious User-Level Payloads
## Possible Attacks

<table>
<thead>
<tr>
<th>Attack Name</th>
<th>Memory Access</th>
<th>Attack Type</th>
<th>Experiment Setup</th>
</tr>
</thead>
<tbody>
<tr>
<td>System corruption attack</td>
<td>-</td>
<td>Denial of service</td>
<td>-</td>
</tr>
<tr>
<td>Bash keyboard buffer attack</td>
<td>Read-Only</td>
<td>Key stroke sniffing</td>
<td>- Dell PE R815</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- AMD Opteron 6128</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- 4 nodes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- Linux kernel 3.6.0</td>
</tr>
<tr>
<td>Bash shell credential object attack</td>
<td>Read-Write</td>
<td>Privilege escalation</td>
<td>- Dell PE R620</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- Intel Xeon E5-2650</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- 2 nodes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- Linux kernel 3.6.0</td>
</tr>
<tr>
<td>Shared library attack</td>
<td>Read-Write</td>
<td>Hidden function (Backdoor)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Overview of Bash Shell Credential Object Attack

Scanning the Fingerprint:
To find the credential object in the VICTIM REGION, an attacker needs to find the fingerprint by READ operation at ATTACK REGION.

- Memory Allocation (`malloc huge-page`)
- Translation of VA to PA (`pagemap` interface)
- PIkit Installation (DRAM mapping table modification)
- Fingerprint Scanning

: Attack Node
: PIkit Attack Node
: Victim Node
Scanning the Fingerprint

**Process Table**

<table>
<thead>
<tr>
<th>PID</th>
<th>PCB</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>n</td>
<td></td>
</tr>
</tbody>
</table>

**Credential Management** (include/linux/cred.h)

```
struct cred {
    uid_t uid;        /* real UID of the task */
    gid_t gid;        /* real GID of the task */
    uid_t suid;       /* saved UID of the task */
    gid_t sgid;       /* saved GID of the task */
    uid_t euid;       /* effective UID of the task */
    gid_t egid;       /* effective GID of the task */
    uid_t fsuid;      /* UID for VFS ops */
    gid_t fsgid;      /* GID for VFS ops */
};
```

1. are known to the attacker (UID & GID)
2. should be within 0xffff880000000000 – 0xffffc7ffffff000000
3. can be found in Symbol Lookup Table (/boot/System.map)
Overview of Bash Shell Credential Object Attack

Memory Allocation (malloc huge-page)

VA

Translation of VA to PA (pagemap interface)

PA

Plkit Installation (DRAM mapping table modification)

Fingerprint Scanning

Found!

Yes

Data Modification

Scanning the Fingerprint:
To find the credential object in the VICTIM REGION, an attacker needs to find the fingerprint by READ operation at ATTACK REGION

Modifying the Data:
If the fingerprint is found, an attacker can overwrite the EUID (or UID) to 0
Modifying the Data

- Once the corresponding address of the credential data structure is determined from the scanning, the attacker can get a root shell by modifying either the `euid` or the `uid` field.

  \[
  \text{movnti} \ $0, \ (\text{Virtual Address})
  \]

- Result

```
[smith@server ~]$ id
uid=500(smith) gid=500(smith) groups=500(smith) context=unconfined_u:unconfined
[smith@server ~]$ id
uid=500(smith) gid=500(smith) euid=0(root) egid=0(root) groups=0(root) linect=0-0:0-0
```
Overview of Bash Shell Credential Object Attack

- Memory Allocation (malloc huge-page)
- Translation of VA to PA (pagemap interface)
- Plkit Installation (DRAM mapping table modification)
- PCB(Bash $) Spraying
- Fingerprint Scanning
- Data Modification

Scanning the Fingerprint:
To find the credential object in the VICTIM REGION, an attacker needs to find the fingerprint by READ operation at ATTACK REGION

Modifying the Data:
If the fingerprint is found, an attacker can overwrite the EUID (or UID) to 0

Spraying the Process Control Block:
An attacker can increase the possibility that the credential data structure is placed in the VICTIM REGION
Overview of Shared Library Attack

- **Attack Node**: PIkit
- **Victim Node**: Any Nodes
- **Any Nodes**: SUID program execution

---

### Overview

**Shared Library Attack**

1. **Malicious Binary Code**: Re-write malicious binary code for the libc function
2. **Translation**: Convert VA to PA (pagemap interface)
3. **PIkit Installation**: SAD & TAD modification
4. **Re-write Binary**: Re-write malicious code for the libc function
5. **Execution**: SUID program execution linked with the libc function
6. **Restore Original**: Restore original binary code for the libc function

---

**Example Code**

```
// dummy.c
void main(void){
    struct passwd *ret;
    int pid = getpid();
    int uid = getuid();
    ret = getpwuid(uid);
    printf("pid : %d
",pid);
    sleep (600);
}
```

```
$ ./dummy &
pid : 7145
```

```
$ gdb dummy 7145
(gdb) p getpwuid
$1 = {<text variable>}
```

```c
getpwuid()
size_t buffersize;
malloc(buffersize);
```

---

**Original Binaries**

```c
getpwuid()
setuid(0);
system("/bin/sh");
```

---

**Modified Binaries**

```c
getpwuid()
setuid(0);
system("/bin/sh");
```

---

**Malicious Binaries**

```c
getpwuid()
setuid(0);
system("/bin/sh");
```

---

**Privilege Escalation**

```
 mike@server ~]$ passwd
```

```
uid=702(mike) gid=702(mike) euid=0(root)
```

---

**System Security Lab**

---

26
Discussions – Possible Solutions
Possible Solutions

❖ Using Existing Features
   – Enable AMD’s LockDramCfg feature (included in “C6 State Management”)
     ▪ In old BIOS, it is disabled by default
   – Similar LockDramCfg feature is not readily available in Intel x86 CPUs.

❖ Software-based Solution
   – DRAM address mapping table monitor
     ▪ Should be protected with a secure platform such as TrustZone, Intel SGX

❖ Hardware-based Solutions
   – Restrict # of entries used to equal # of nodes in the system
     ▪ Does not completely remove the possibility of the PKit
   – Design the DRAM address mapping table entries as write-once memory-mapped registers
     ▪ Flexibility issues for CPU Hotplug or memory Hotplug
Conclusion

- PIkit – a kernel-independent processor-interconnect rootkit
  - Exploits vulnerable hardware features in Processor-Interconnect
  - Kernel Independent (No code injection or code modification to the kernel)
    - Enables malicious activities to be carried out with only user-level code

- Four different attack scenarios
  - Bash shell credential object attack, Bash keyboard buffer attack, Shared library attack

- PIkit is a very stealthy rootkit
  - Payloads consist of only read/write memory operations
Thank you! Questions?

zemisolsol@kaist.ac.kr