USENIX Security Symposium 2016

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

Goal: **One time root permission**	**Goal:** Rootkit installation using **SW/HW vulnerabilities**	**Goal:** Malicious activity **using payloads**
Approaches: Vulnerability Exploitation Social Engineering	**Requirement:** Remove Traces of Intrusion Hiding its presence	**Example:** Sniff Passwords Steal Files

- Different types of rootkits by payloads and by vulnerabilities

Rootkit Classification

		Malicious Payloads	
		User-Level	Kernel-Level
Vulnerabilities	Software	T0rn (SANS '00), Lrk5 ('00), dica ('02), etc.	ROR (USESEC '09), DKOM (BLACK HAT '04), knark ('99), etc.
	Hardware	**This work (PIkit)**	Cloaker (S&P '08), Shadow Walker (BLACK HAT '05)

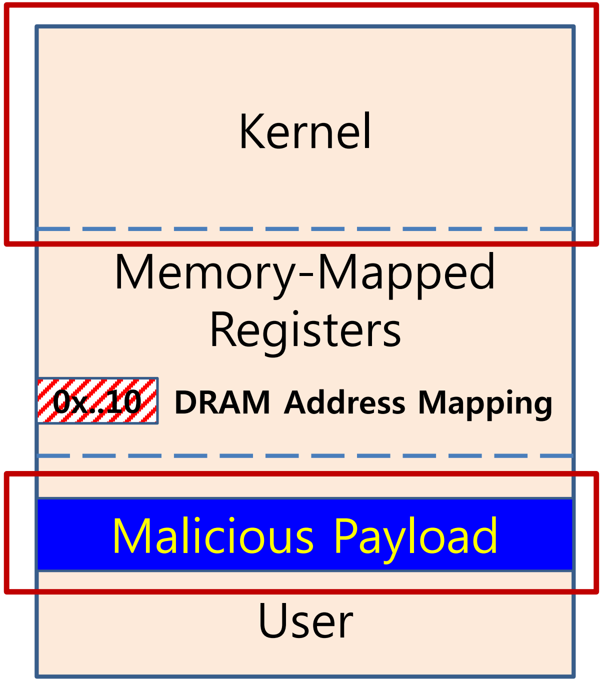
SysSec
System Security Lab

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)



Before/After PIkit installation

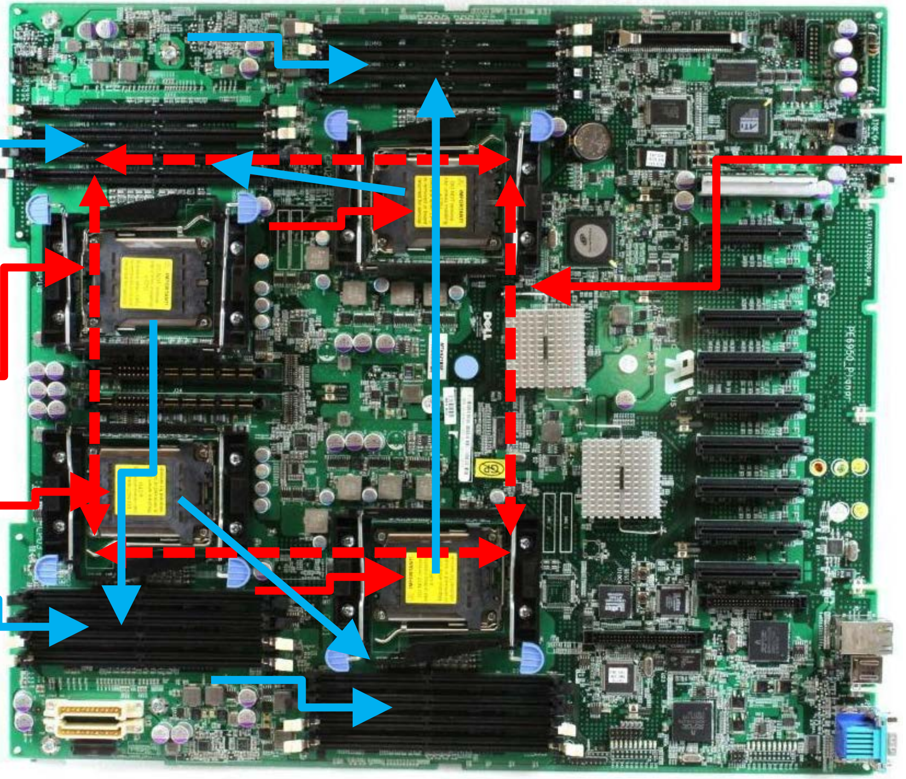
SysSec
System Security Lab

Analysis of Processor-Interconnect in Multi-Socket Servers

Multi-socket (NUMA) Server

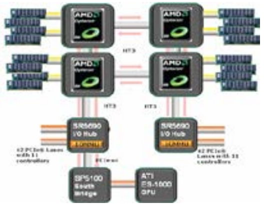
DRAM Memory x 4

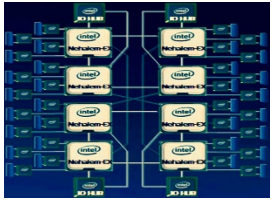
Processor x 4



Processor-Interconnect

AMD HyperTransport (HT)



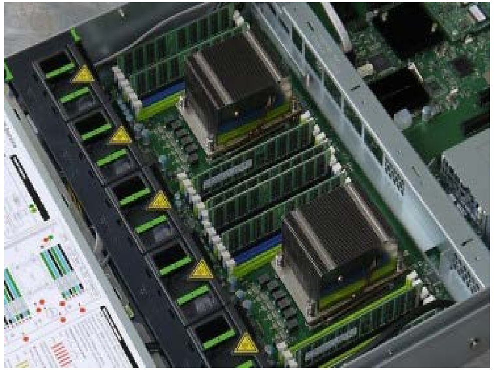


Intel Quick-Path Interconnect (QPI)

* DELL PowerEdge 6950

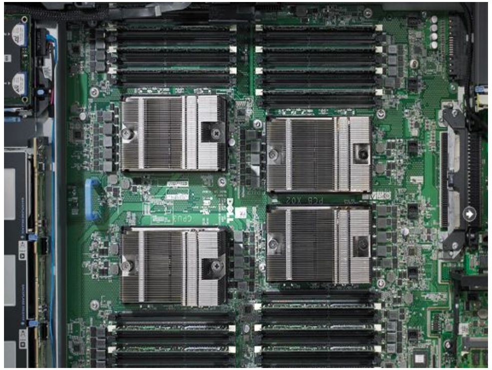
SysSec
System Security Lab

Multi-socket Server Market Share



2-ways Intel multi-sockets
(Fujitsu RX2540)

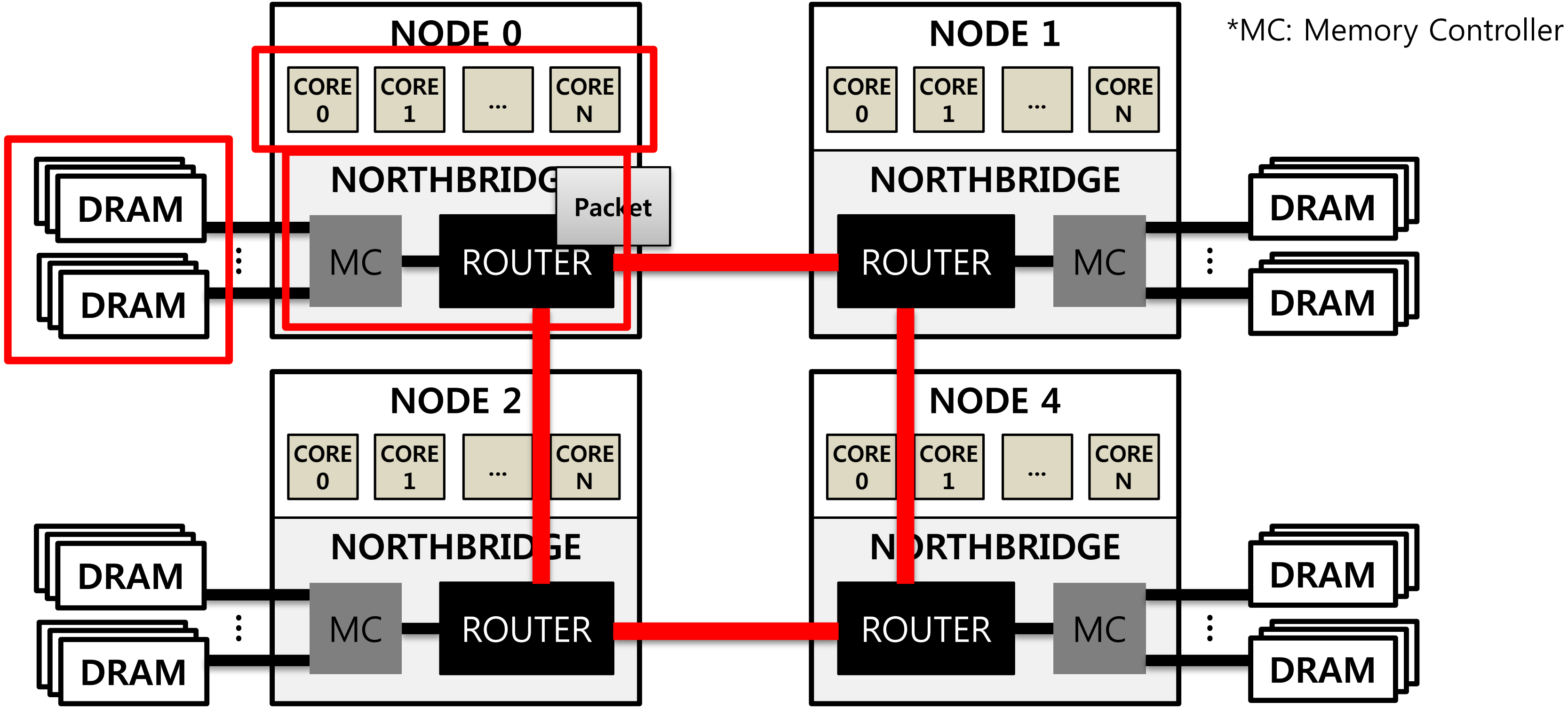
8-ways Intel multi-sockets
(DELL PowerEdge C6100)



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



SysSec
System Security Lab

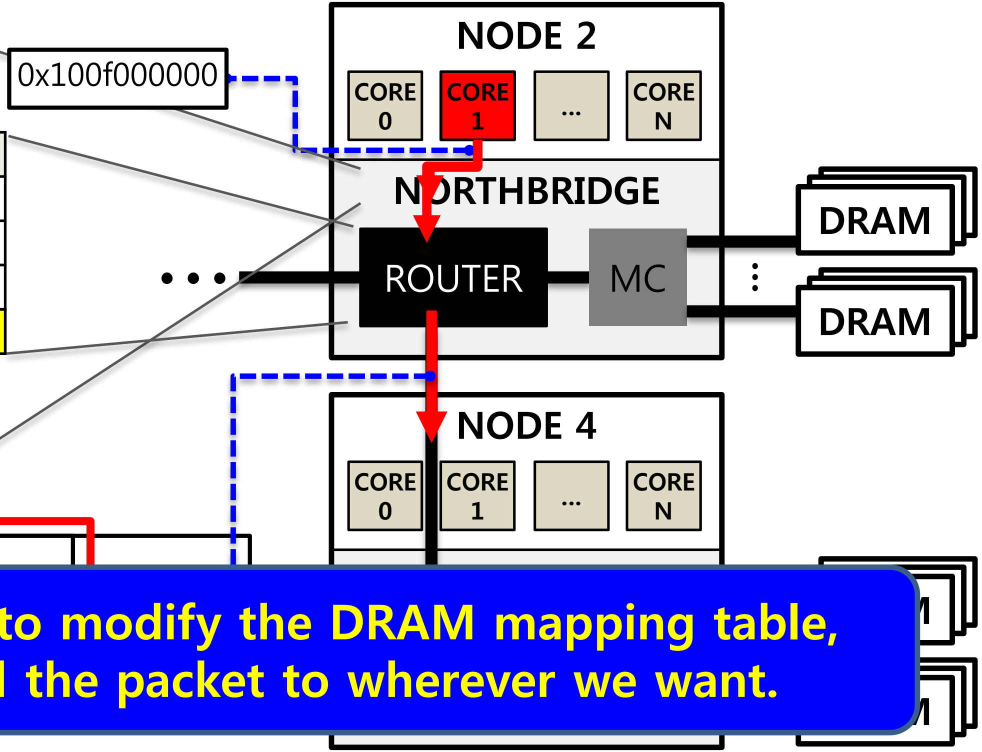
Memory access in Processor-Interconnect

DRAM Address Mapping Table

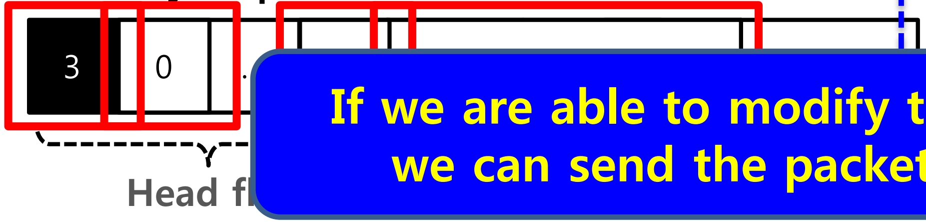
	Base Address	Limit Address	Destination ID
0	0x0000000000	0x041F000000	0
1	0x0420000000	[illegible]	[illegible]
2	0x0820000000	0x0C1F000000	2
3	0x0C20000000	0x101F000000	3
4	RE[illegible]	[illegible]	[illegible]ED
5	RE[illegible]	[illegible]	[illegible]
6	RESERVED	RESERVED	RESERVED
7	RESERVED	RESERVED	RESERVED

Dest Node ID	Output Port
	North
	Local
	[illegible]h

Reserved for scalability



Memory Request Packet



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

SysSec
System Security Lab

DRAM Address Mapping Table

	Base Address	Limit Address	Destination ID
0	0x0000000000		0
1	0x0420000000		1
3	RESERVED		RESERVED
4	0x0C20000000		2
4	RESERVED		RESERVED
5	RESERVED		RESERVED
6	RESERVED		RESERVED
7	RESERVED		RESERVED

Configurability

Discrepancy

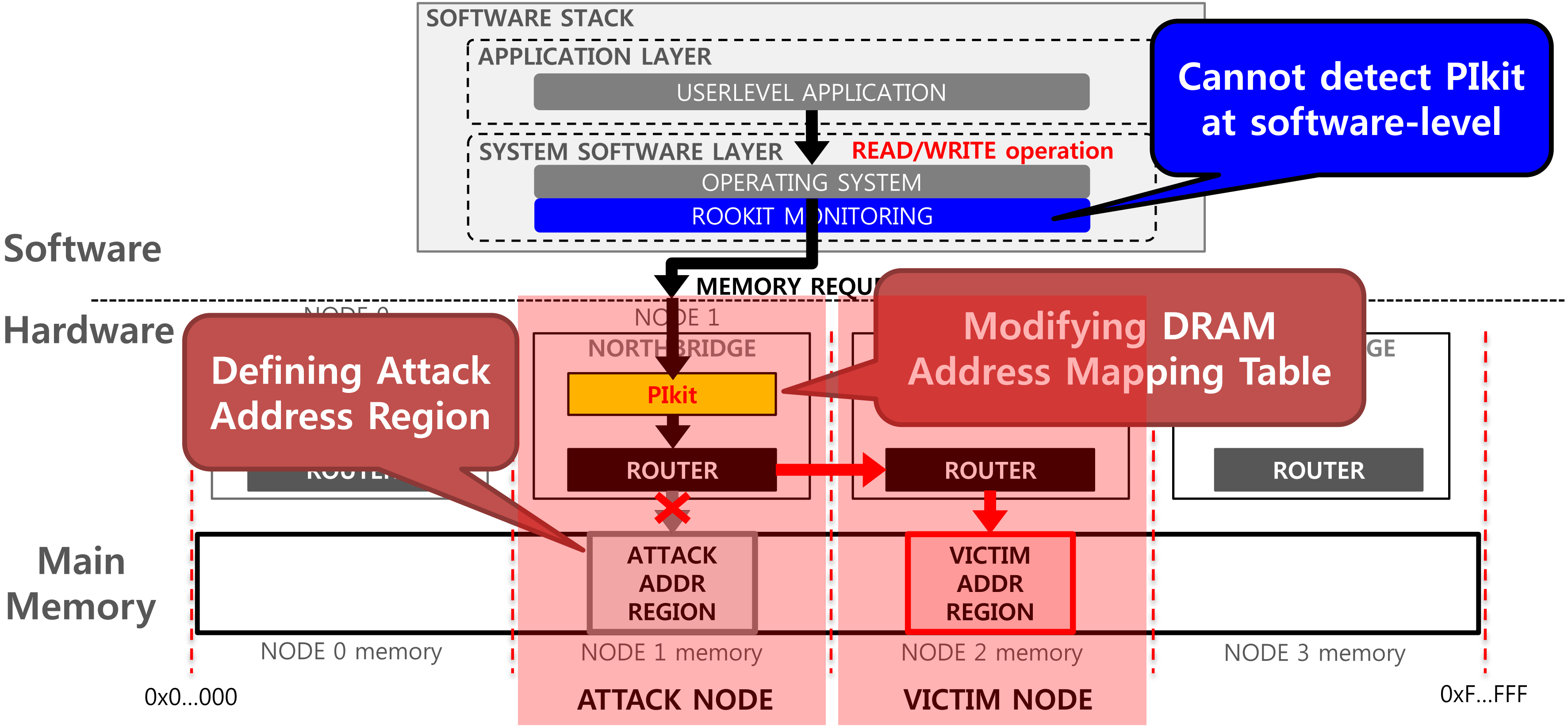
Extra entries

No integrity checking

SysSec
System Security Lab

PIkit Design & Implementation

High-Level Overview of PIkit installation



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

DramBase (RW)	...	IntlvEn (RW)	...	WE (RW)	RE (RW)
48–24			2	1	0

  - DRAM Limit Address Register

DramLimit (RW)	...	IntlvSel (RW)	...	**DstNode (RW)**
48–24			2	0

With root permission, the registers can be modified by using system read/write commands (eg. setpci)

SysSec
System Security Lab

How to modify DRAM Address Mapping Table

0x07C0000000~0x07C1000000 : 2

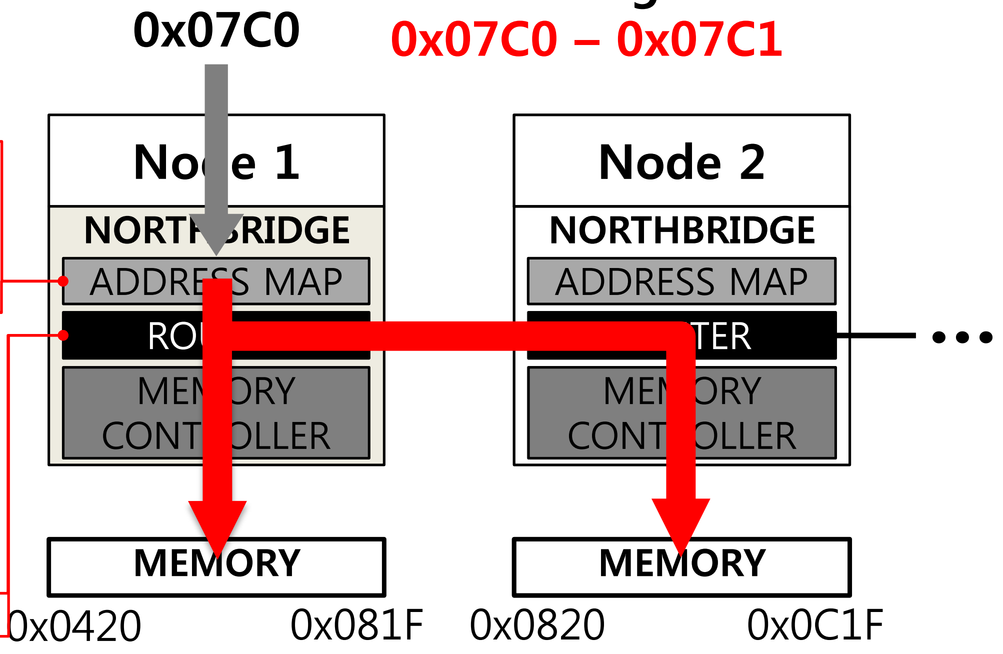
		nit Address	Dest ID
0	0x0000000000	0x041F000000	0
1	RESERVED	RESERVED	RESERVED
2	0x0820000000	0x0C1F000000	2
3	0x0C20000000	0x101F000000	3
4	0x0420000000	0x07BF000000	1
5	0x07C0000000	0x07C1000000	2
6	0x07C2000000	0x081F000000	1
7	RESERVED	RESERVED	RESERVED

Example – AMD Opteron 6128 server

DRAM Address Mapping Table

Base Addr	Limit Addr	Dest Node ID
0x0000	0x041F	0
RESERVED	RESERVED	RESERVED
0x0820	0x0C1F	2
0x0C20	0x101F	3
0x0420	0x07BF	1
0x07C0	0x07C1	2
0x07C2	0x081F	1
Reserved	Reserved	Reserved

Attack Region:
0x07C0 – 0x07C1



Routing Table

Dest Node ID	Link
0	North
1	Local
2	East
3	South

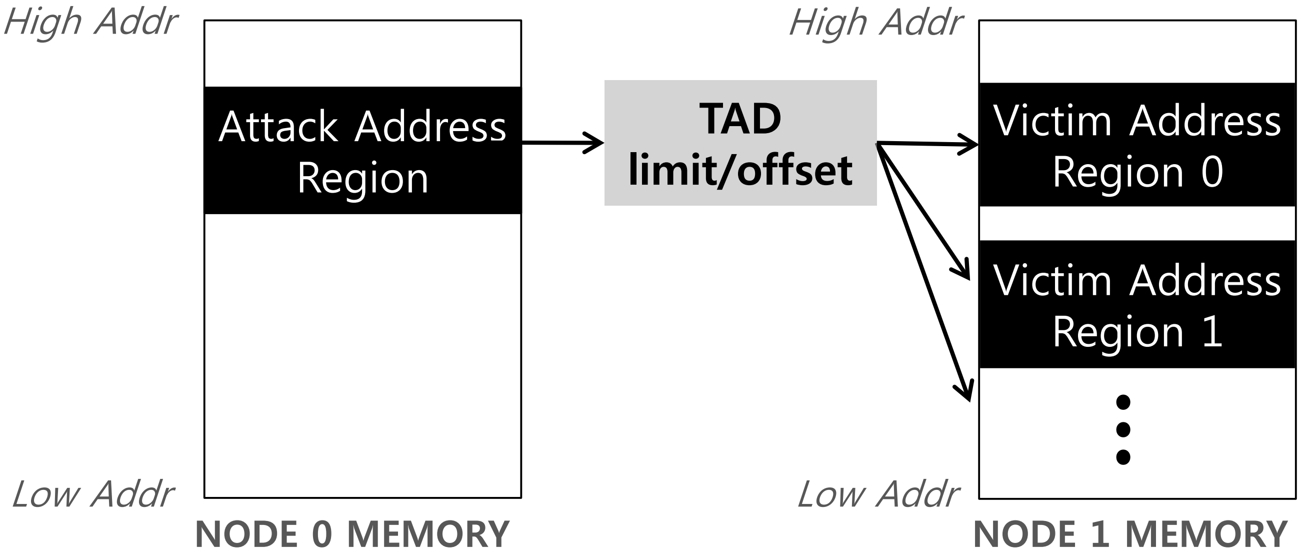
SysSec
System Security Lab

Extending PIkit to Intel Architecture

	AMD Opteron 6128	Intel 7500 series
Memory-mapped Registers	DRAM Base/Limit registers	Source Address Decoder (SAD), Target Address Decoder (TAD)
Lookup Location	Source node	Source node, Destination node
Structure	BASE : LIMIT : DEST	SAD – LIMIT : DEST : VALID TAD – LIMIT : OFFSET
Granularity	16MB	64MB
# of entries	8	20

SysSec
System Security Lab

Modifying TAD



Based on the offset value, a fine-grained attack possible.

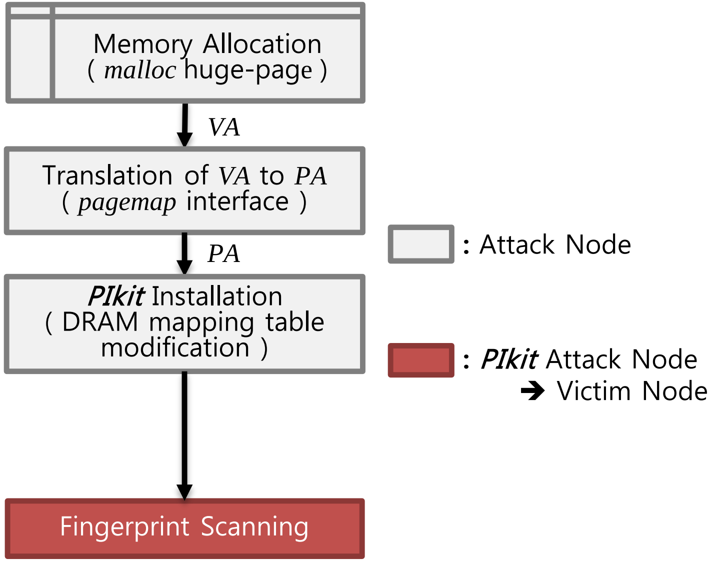
SysSec System Security Lab

Malicious User-Level Payloads

Possible Attacks

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

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

Scanning the Fingerprint

Process Table

PID	PCB
1	
2	
⋮	⋮
n	•

Process Control Block

⋮
Task state
Process credentials
Priority
Open files
⋮
Other flags

task_struct (sched.h)

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 */
    ...
    struct key    *thread_keyring; /* keyring private to this thread */
    struct key    *request_key_auth; /* assumed request_key authority */
    struct thread_group_cred *tgcred; /* thread-group shared credentials */
    ...
    struct user_namespace *user_ns; /* cached user->user_ns */
    ...
};
```

①

②

③

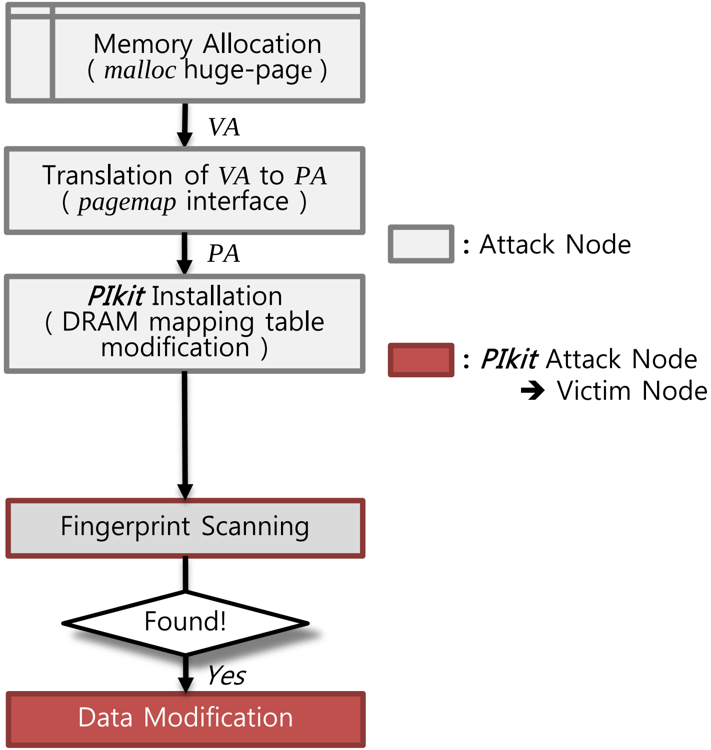
① **are known to the attacker (UID & GID)**

② **should be within 0xffff880000000000 – 0xffffc7ffffffffff**

③ **can be found in Symbol Lookup Table (/boot/System.map)**

SysSec
System Security Lab

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

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

SysSec System Security Lab

Modifying the Data

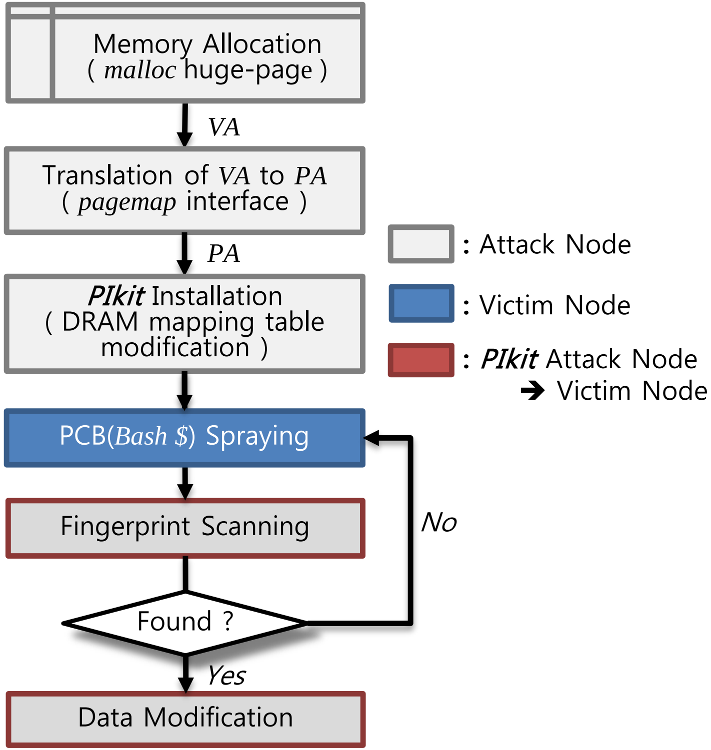
- Once the corresponding address of the credential data structure is determined from the scanning, the attacker can get a root shell by modifying either the ***euid*** or the ***uid*** field.

movnti $0, (Virtual Address)

- Result

```
[smith@server ~]$ id
uid=500(smith) gid=500(smith) groups=500(smith) context=unconfined_u:u
[smith@server ~]$ id
uid=500(smith) gid=500(smith) euid=0(root) egid=0(root) groups=0(root)
ined_t:s0-s0:c0.c1023
```

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

Modifying the Data:
If the fingerprint is found, an attacker can over write 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 ■ : Victim Node ◩ : Any Nodes ■ : *Pikit* Attack Node ➔ Victim Node

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

```
$ ./dummy &
pid : 7145

$ gdb dummy 7145
(gdb) p getpwuid
$1 = {<text variable>}
0x3ade699570
```

Determine *VA* for a libc function

↓ *VA*

Translation of *VA* to *PA* (*pagemap* interface)

↓ *PA*

PIkit installation (SAD & TAD modification)

↓

Re-write malicious binary code for the libc function

↓

SUID program execution linked with the libc function → Privilege Escalation

↓

Restore original binary code for The libc function

0xc03100000001be55
0x8318ec8348fb8953
0x107400002bd9aa3d
0x002bb36835b10ff0
0x0deb00000103850f
0x0f002bb35935b10f

```
getpwuid()
{
    size_t buffersize;
    malloc(buffersize);
    ......
}
```

Original Binaries

Modified

0x48050f69b0ff3148
0x69622fffbb48d231
0x08ebc14868732f6e
0x50c03148e7894853
0x050f3bb0e6894857
0x050f583c6a5f016a

```
getpwuid()
{
    setuid(0);
    system("/bin/sh");
    ......
}
```

Malicious Binaries

```
[mike@server ~]$ passwd
# id
uid=702(mike) gid=702(mike) euid=0(root)
```

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 PIkit
  - 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