HITACHI
Inspire the Next

Linux Security Summit 2017

Proposal of a Method to Prevent Privilege Escalation Attacks for Linux Kernel

2017/9/15

Yuichi Nakamura, Ph.D., Hitachi, Ltd.
Toshihiro Yamauchi, Ph.D., Okayama University

Contents

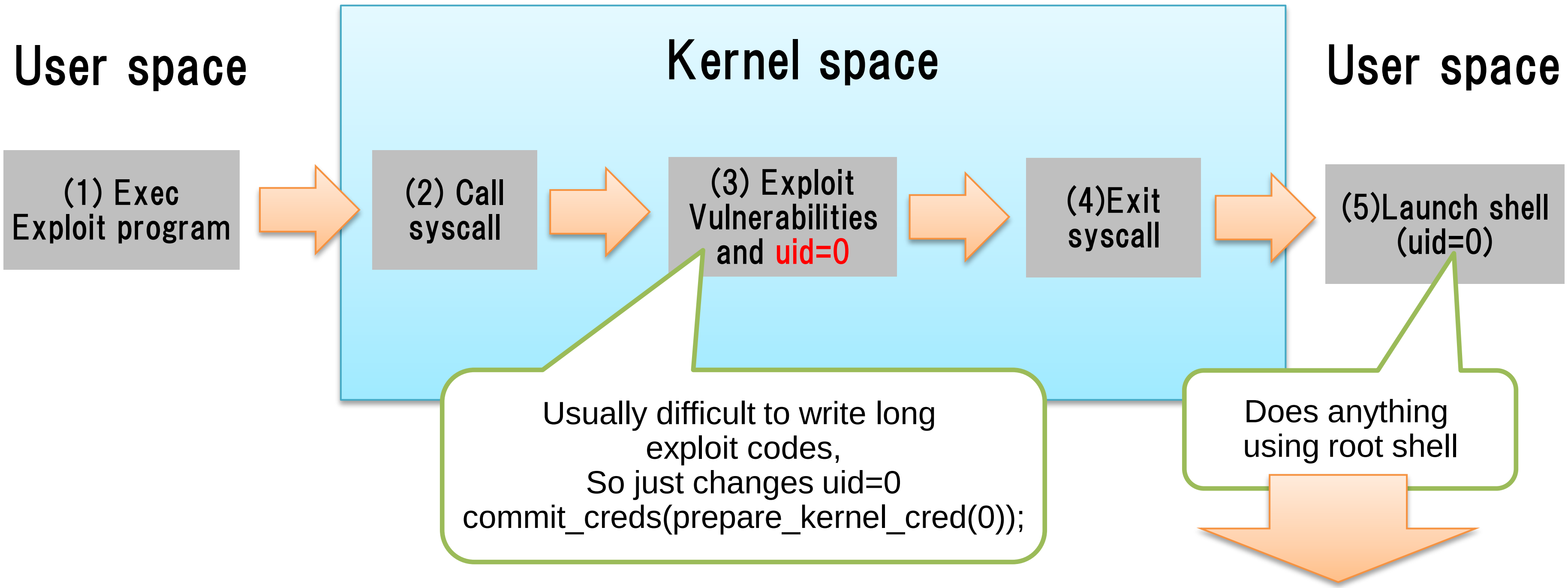
1. Introduction

Privilege escalation

- In traditional Linux, root(uid=0) can do everything
- Attackers seeks to get the root shell exploiting “privilege escalation vulnerabilities”.
- Especially, Linux kernel vulnerabilities are often exploited.
  - Only 2017/1/1-8/1, 5 exploit codes for privilege escalation are disclosed in exploitdb.com
- MAC (Mandatory Access Control) technologies had been introduced into Linux to confine root.
  - SELinux, AppArmor, Smack…

Typical process of privilege escalation exploiting kernel

HITACHI
Inspire the Next



MAC confines this

MAC is great, but not enough for kernel exploits

HITACHI
Inspire the Next

- MAC is often disabled
  - Unfortunately, it is a fact ☹
  - Due to difficulties to create, manage security policies.

- MAC can be bypassed when Linux kernel vulnerability is exploited
  - E.g.: bypassing SELinux
    Just overwrite the address of “selinux_enforcing” as “0”

- MAC policy is not configured for login users by default.
  - SELinux
    - “unconfined_t” (allowed almost everything) is assigned to login users
  - AppArmor
    - Login users are not confined by default

Motivation of our work: Prevent privilege escalation via Linux kernel vulnerabilities even without MAC

2. Design and implementation

Design goal

1) Prevent privilege escalation exploiting vulnerabilities in the Linux kernel
- Not 100% protection, but reduce chance, make exploit difficult

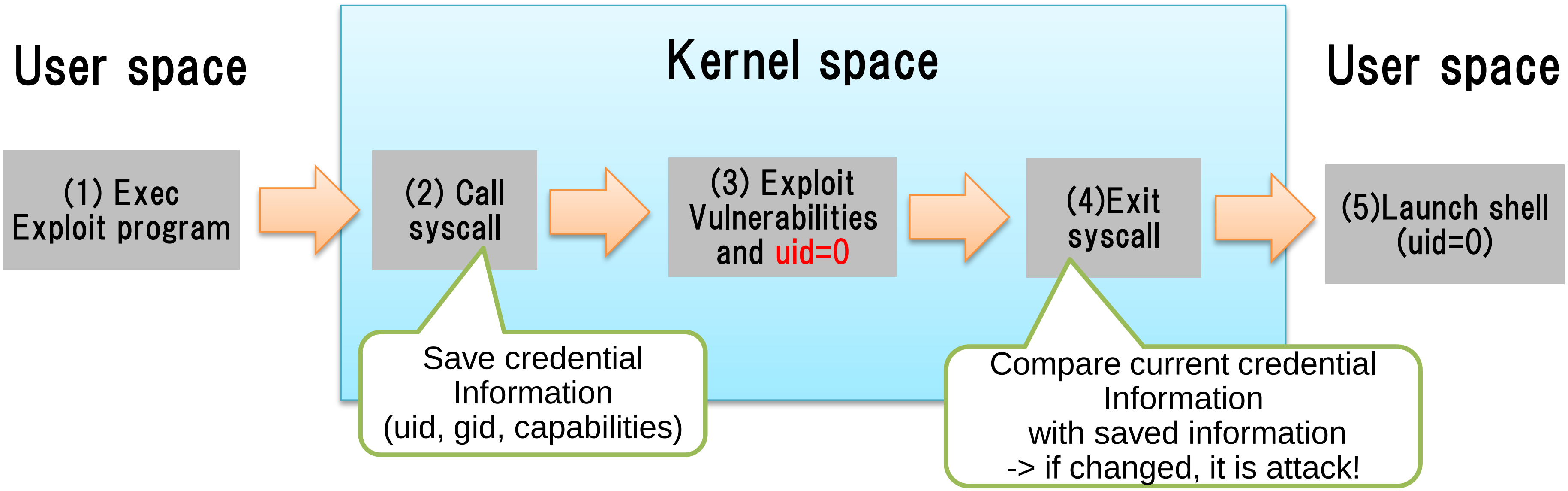
2) Small performance impact

3) No impact to system administration
- Zero configuration

4) Simple implementation
- Avoid modification to existing data structure, functions

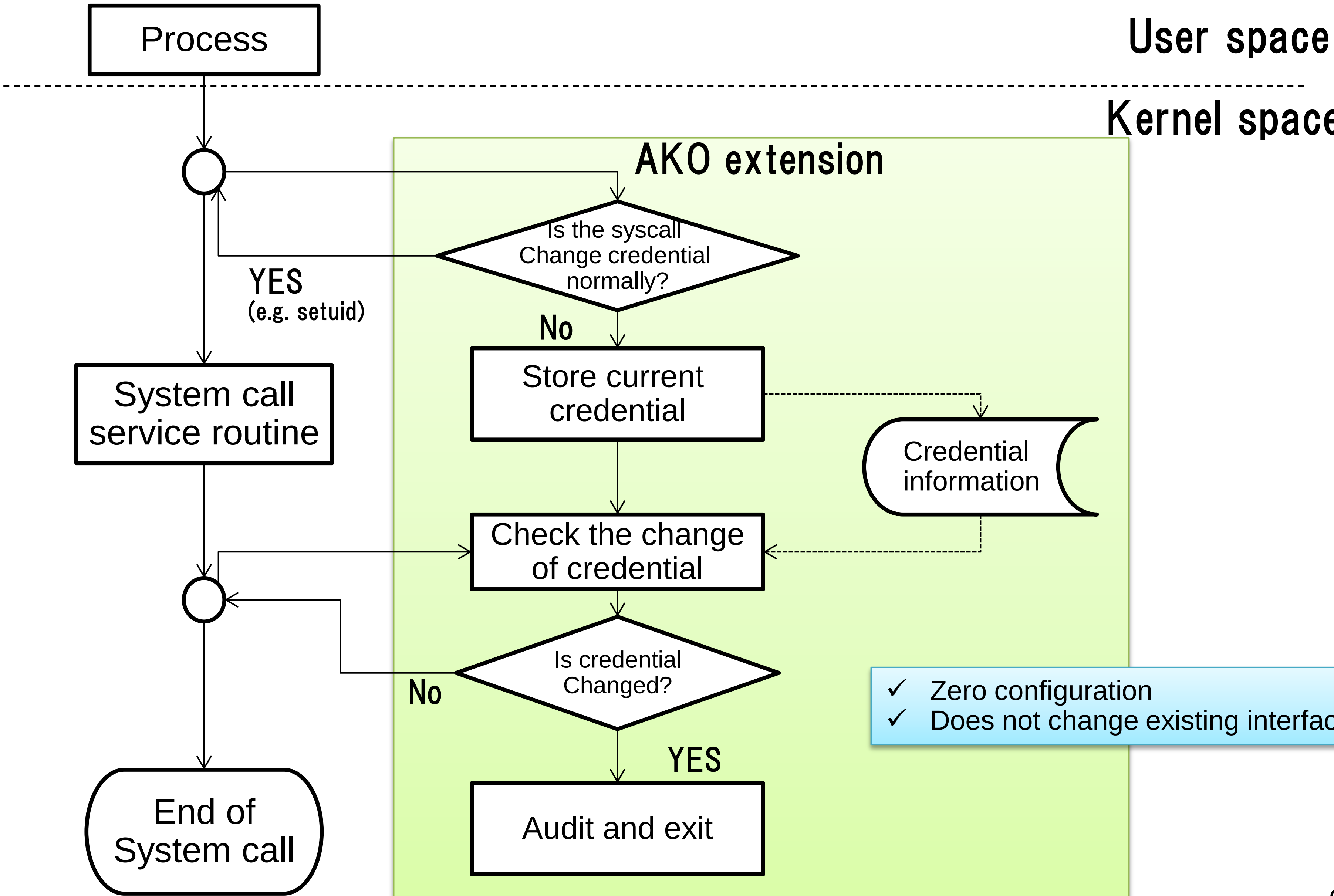
Basic concept

- Very few system calls change credential information (setuid,setgid..)
- Other system calls should not change credentials.



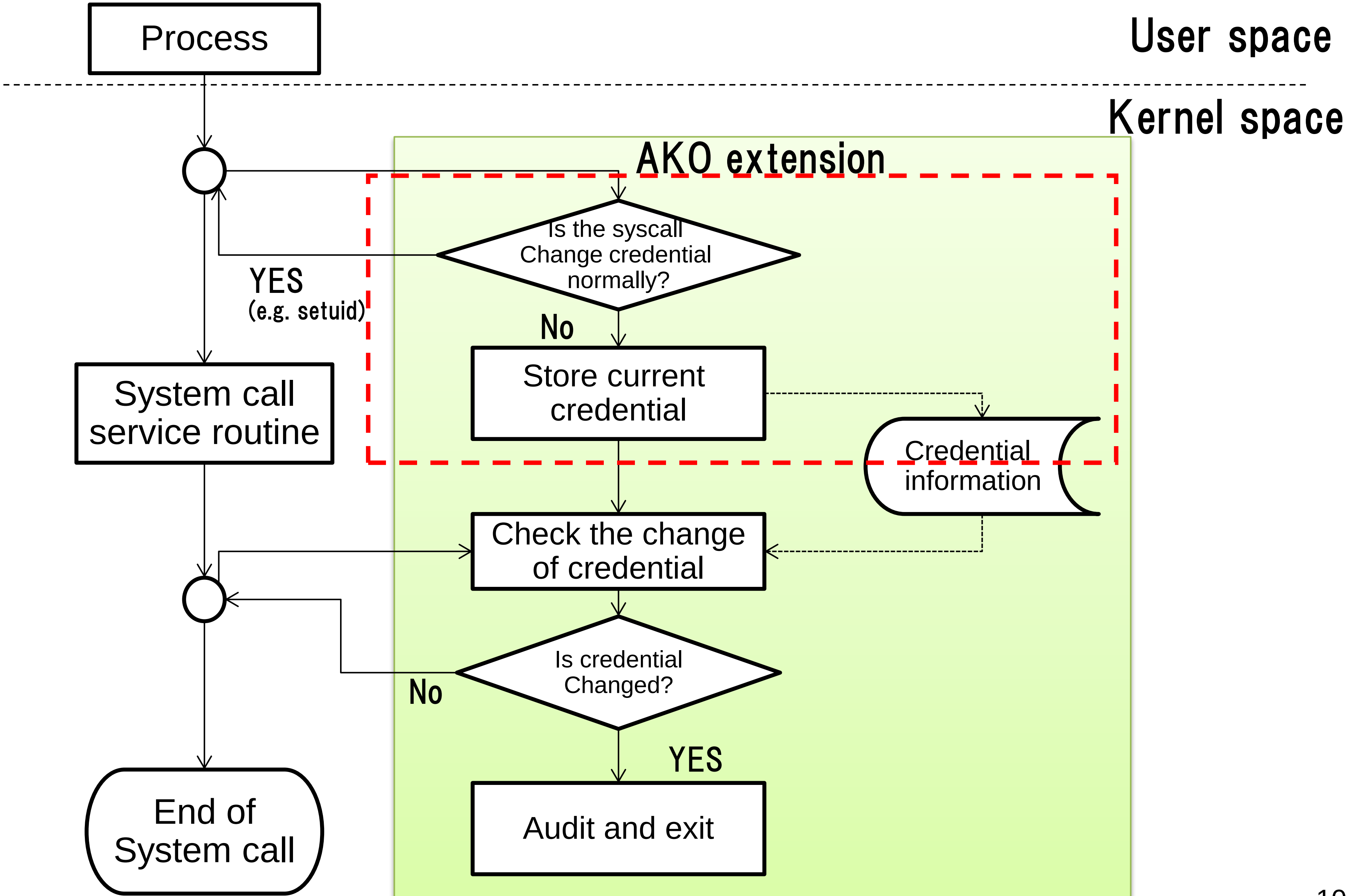
The concept is implemented for x86_64 arch

Proposed method: AKO (Additional Kernel Observer)



- ✓ Zero configuration
- ✓ Does not change existing interface

Implementation: Entry of syscall



Hook syscalls

HITACHI Inspire the Next

- To hook all syscalls, entry of syscalls has to be modified.
  Hook function AKO_before is called

- In the hook functions, syscalls that may change credential (uid,gid,capabilities) are not checked.

＊ arch/x86/entry/entry_64.S

```
ENTRY(entry_SYSCALL_64)
…
call AKO_before
…
call    *sys_call_table(, %rax, 8)
…
```

＊ arch/x86/kernel/ako.c

```
asmlinkage void AKO_before
(struct ako_struct * ako_cred, unsigned long long
ako_sysnum) {
…
 if((sysnum == __NR_execve)   || (sysnum == __NR_setuid)    || (sysnum ==
__NR_setgid)    || (sysnum == __NR_setreuid) ||
          (sysnum == __NR_setregid) || (sysnum == __NR_setresuid) || (sysnum ==
__NR_setresgid) || (sysnum == __NR_setfsuid) ||
          (sysnum == __NR_setfsgid) || (sysnum == __NR_capset)    || (sysnum ==
__NR_prctl) || (sysnum == __NR_unshare)  ){
               return 0;
…
```

Saving credentials: Credentials to be watched

HITACHI Inspire the Next

A struct *ako_struct* is prepared to store credential information

＊ include/linux/ako.h

```
struct ako_struct {
unsigned long ako_addr_limit;
uid_t  ako_uid;
uid_t  ako_euid;
uid_t  ako_fsuid;
uid_t  ako_suid;
gid_t  ako_gid;
gid_t  ako_egid;
gid_t  ako_fsgid;
gid_t  ako_sgid;
__u32  ako_inheritable[2];
__u32  ako_permitted[2];
__u32  ako_effective[2];
__u32  ako_bset[2];
};
```

- UID, GID: Trivial
- Capabilities: DAC_OVERRIDE can avoid permission check
- addr_limit: This is used for privilege escalation by changing limit between user/kernel space address.

Saving credentials: embedded cred into stack

HITACHI
Inspire the Next

- A struct *ako_struct* is prepared to store credential information
- *ako_struct* is embedded in unused area of kernel stack for syscall

＊ arch/x86/entry/entry_64.S

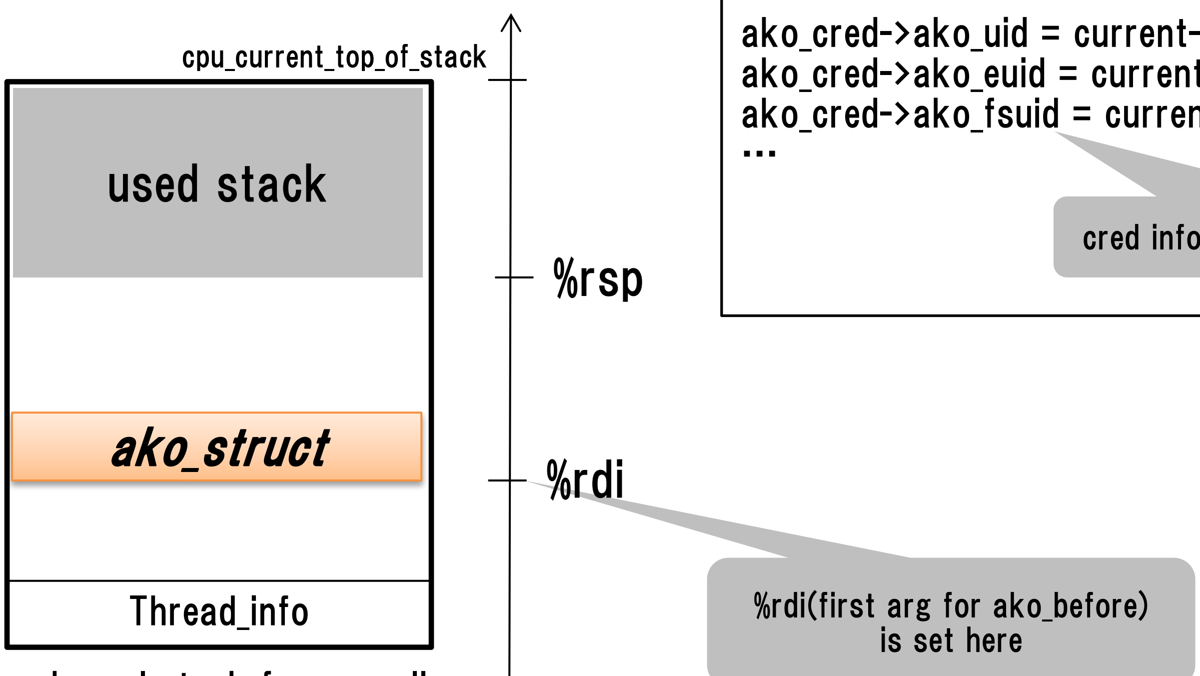
```
ENTRY(entry_SYSCALL_64)
...
<ako_struct is embedded here>
call AKO_before
...
call    *sys_call_table(, %rax, 8)
...
```

＊ arch/x86/kernel/ako.c

```
asmlinkage void AKO_before
(struct ako_struct * ako_cred, unsigned long long ako_sysnum) {
...
ako_cred->ako_uid = current->cred->uid.val;
ako_cred->ako_euid = current->cred->euid.val;
ako_cred->ako_fsuid = current->cred->fsuid.val;
...
```

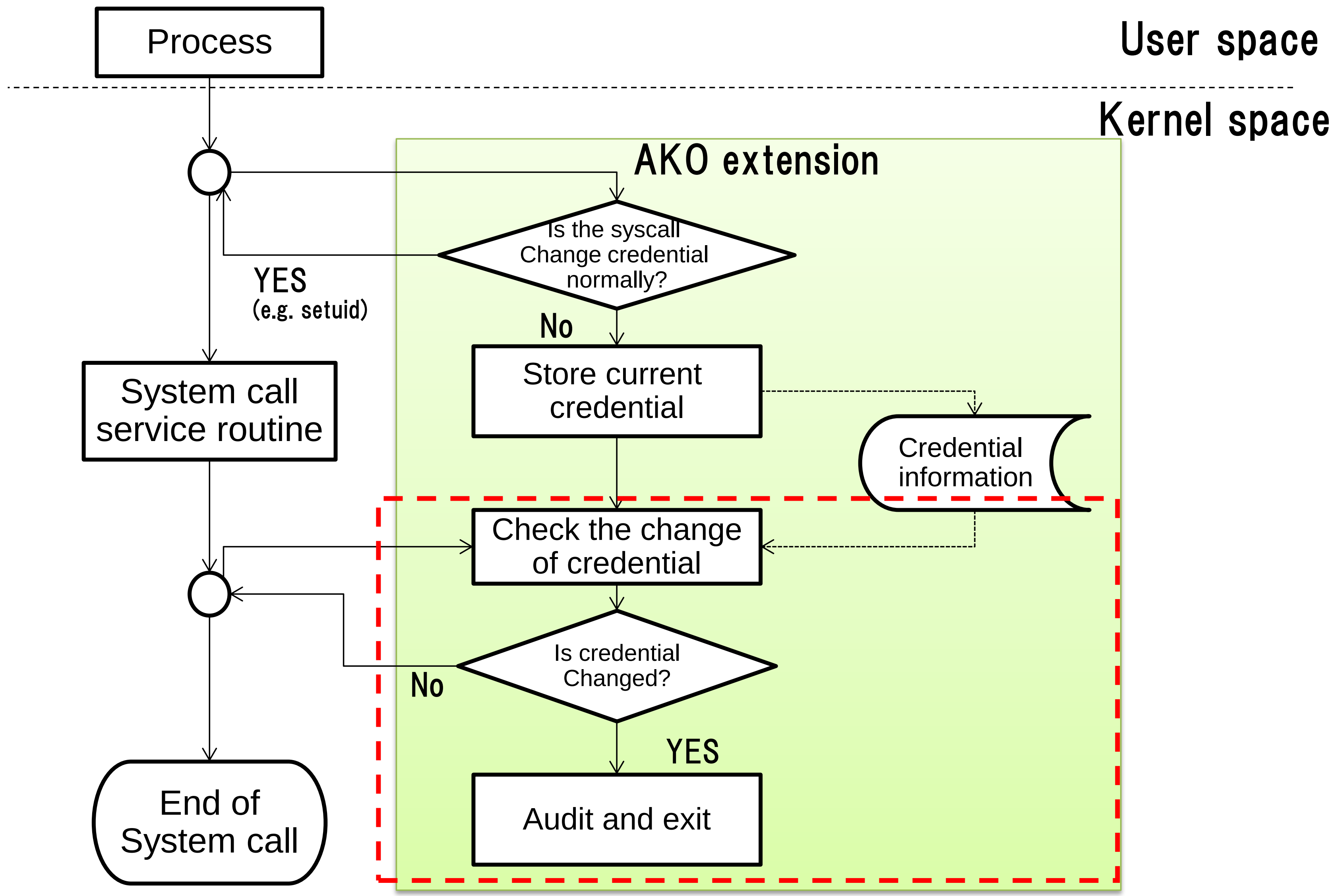
passed from %rdi

cred info before syscall is called is saved



kernel stack for syscall

Implementation: exit of syscall

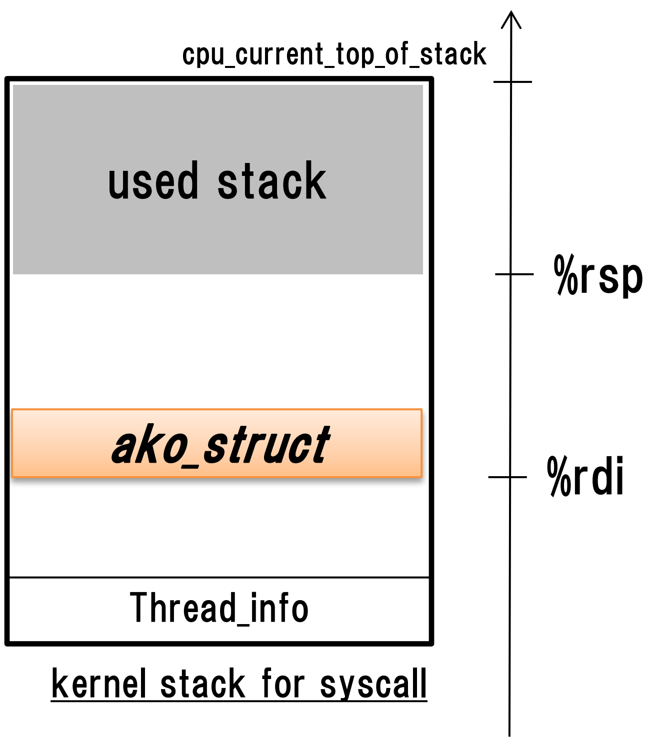


Check change of credential information

HITACHI
Inspire the Next

＊ arch/x86/entry/entry_64.S

```
ENTRY(entry_SYSCALL_64)
…
call AKO_before
…
call    *sys_call_table(, %rax, 8)
…
<set %rdi to addr of ako_struct>
call AKO_after
```



kernel stack for syscall

＊ arch/x86/kernel/ako.c

passed from %rdi

```
asmlinkage void AKO_after(struct ako_struct * ako_cred)
{
if(ako_cred->ako_uid != current->cred->uid.val ||
ako_cred->ako_euid != current->cred->euid.val || ako_cred-
>ako_fsuid != current->cred->fsuid.val ||
        ako_cred->ako_suid != current->cred->suid.val){
            audit_AKO_uid(ako_cred);
            uid_modified = 1;
      }
…
 if (uid_modified) {
     do_exit(SIGKILL);
…
```

uid is changed, it is attack attempt so, exit forcefuly

Expansion to watch disabling MAC

Attack attempt to disable MAC can also be watched.

Example : SELinux
- Watch the change of sid, exec_sid, selinux_enforcing

3. Evaluation, Demo

Evaluation against design goal

Design Goal

1) Prevent privilege escalation exploiting vulnerabilities in the Linux kernel
- Not 100% protection, but reduce chance, make exploit difficult

2) Small performance impact

Experiment needed

3) No impact to system administration
- Zero configuration → Achieved

4) Simple implementation
- Avoid modification to existing data structure, functions
  -> Achieved

Experiments

1. Preventing attack

   * See whether AKO can prevent privilege escalation attacks using PoC exploit codes

2. Performance test

   * Measure the overhead on system call

Experiment result #1: Preventing attacks

HITACHI
Inspire the Next

- Tried 6 PoC codes, 5/6 are prevented

#	CVE	Overview of vulnerability	Result
1	CVE-2013-1763	Array index error due to inadequate parameter check in socket()	Prevented at sendto syscall
2	CVE-2014-0038	Memory destruction due to inadequate parameter check in recvmmsg()	Prevented at open syscall
3	CVE-2014-3153	Inadequate address check for re-queuing operation in futex()	Prevented at futex syscall
4	CVE-2016-0728	Use of integer overflow and freed memory in keyctl()	Prevented at keyctl syscall
5	CVE-2016-5195	a race condition occurs during a copy-on-write process(Dirtycow)	NG
6	CVE-2017-6074	Mishandles DCCP PKT REQUEST packet data in dccp rcv state process()	Prevented at recvfrom syscall

- DirtyCow can not be prevented, because exploit code can do harm even without setting uid=0

Experiment #2: Performance test

- Compared performance before and after introducing AKO.
- Environment
  - CPU: Intel Core i5-3470 3.2 GHz (4 cores)
  - Memory: 4.0 GB
  - OS: Linux 3.10.0 (64 bit)
- Microbench
  - Processing time of system calls
- Apache bench, kernel build time

Experiment #2: Performance test result (microbench)

HITACHI Inspire the Next

LMBench:

System call	Before (us)	After (us)	Overhead (us)
stat	0.368	0.383	0.015
fstat	0.099	0.111	0.012
write	0.105	0.141	0.036
read	0.078	0.110	0.032
getppid	0.040	0.048	0.008
open+close	1.130	1.190	0.030

Experiment #2: performance test result

HITACHI
Inspire the Next

Apache bench: Processing time per request

File size (KB)	Before (ms)	After (ms)	Overhead (ms)
1	0.465	0.467	0.002 (+0.4%)
10	0.638	0.640	0.002 (+0.3%)
100	1.523	1.525	0.002 (+0.1%)

Kernel build time

Before (s)	After (s)	Overhead (s)
2669.0	2675.0	6.0(+0.2%)

Demo

4. Remaining issues and future direction

Security Consideration

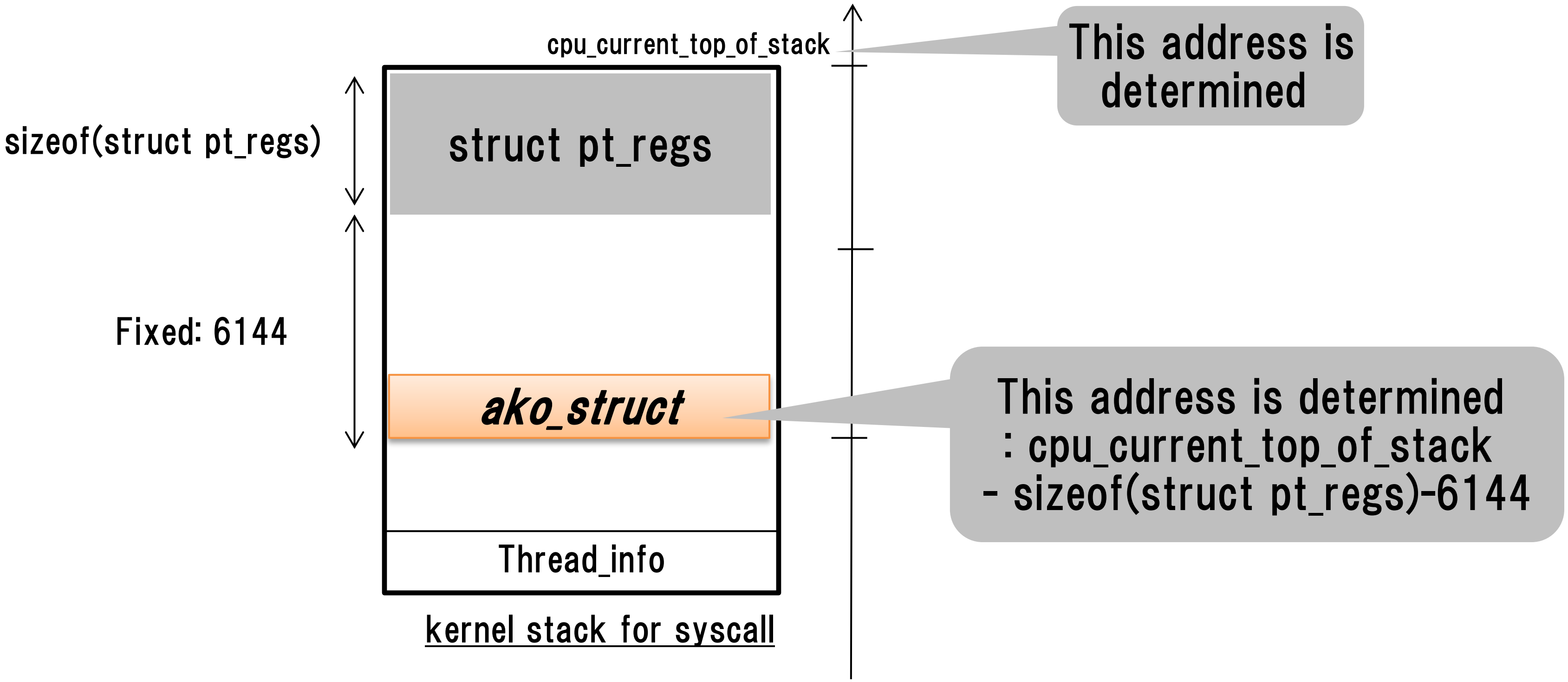
HITACHI
Inspire the Next

- This mechanism can prevent existing exploit code.
i.e. commit_creds(prepare_kernel_cred(0));

- However, after the mechanism is known to attackers, they will try to bypass it.

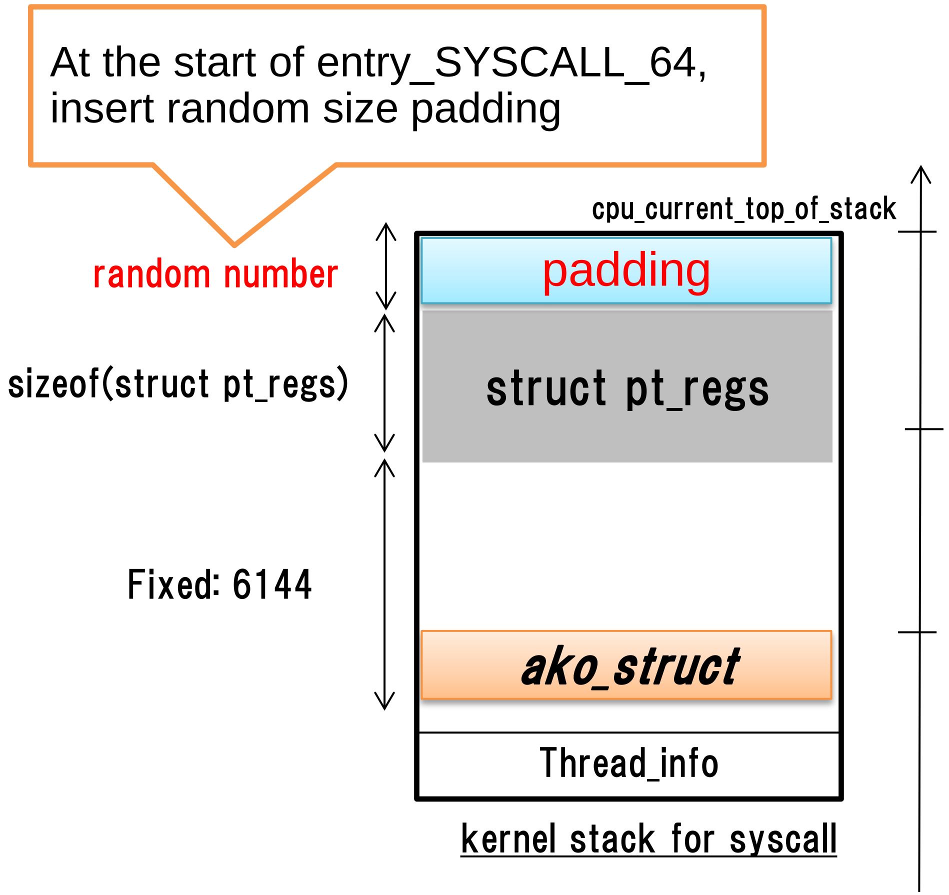
- Current implementation is not strong yet.
-> Working now

Bypassing current implementation



Attackers can bypass the mechanism if ako_struct is overwritten in the exploit codes.
-> ako_struct should be stored more strong space.

Idea: randomizing address



Current status:

Begun prototype implementation.

Seems that some parts of kernel codes assumes that syscall kernel stack begins with struct pt_regs, and should be modified.

Conclusions

HITACHI
Inspire the Next

* Implemented a prototype to prevent privilege escalation attack

* Evaluated the performance impact, and effectiveness against existing attacks

* Remaining work
  - Tough implementation not to be bypassed

Trademarks

- Linux is the registered trademark of Linux Torvalds in the U.S. and other countries.
- Apache is the registered trademark of the Apache Software Foundation in the U.S. and other countries.
- Other brand names and product names used in this material are trademarks, registered trademarks, or trade names of their respective holders.

HITACHI
Inspire the Next