KDRM: Kernel Data Relocation Mechanism to Mitigate Privilege Escalation Attack (Short Paper)

NSS 2023, Session 7: System and Hardware Security, 2023.8.17

Hiroki KUZUNO[a], **Toshihiro YAMAUCHI**[b],
[a]*Graduate School of Engineering,*
Kobe University, Japan
[b]*Faculty of Environmental, Life, Natural Science and Technology,*
Okayama University, Japan

Outline

- Summary and Result
- Motivation and Goal
  - To make the countermeasure mechanism against kernel vulnerability
- The detail of kernel vulnerability attack and Related Work
  - Memory randomization researches for kernel address space
- Problem, Threat Model, and Contribution
- Approach: KDRM (Kernel Data Relocation Mechanism)
  - The design of dynamically replacing credential information against privilege escalation
- KDRM Implementation
  - The software relocation of credential information for the latest Linux kernel
- Evaluation
  - Mitigation result of privilege escalation at the kernel layer
  - Overhead and attack complexity
- Discussion and Conclusion

Summary and Result

- Background and Motivation
  - OS kernel (kernel) vulnerability has become a huge threat to the system security
  - Adversary exploits the kernel vulnerability to compromise the credential management
    - It is an important topic to enhance the kernel resilience against the kernel attack

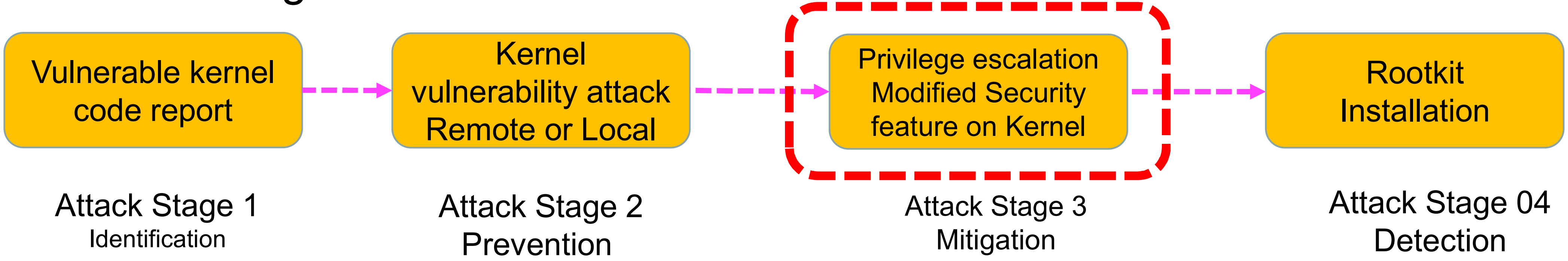
- Approach
  - The purpose of KDRM: Kernel Data Relocation Mechanism
    - It can mitigate a kernel attack threat (e.g., memory corruption)
  - The mechanism relocates the credential information for the running kernel
    - The research tries to make the PoC of kernel data position of kernel memory
    - It tries to achieve the countermeasure of memory corruption attack at the kernel layer
- Results
  - The kernel vulnerability attack failed on the Linux with KDRM
  - Overhead1: LMbench shows that 102.88%-149.67%
  - Overhead2: UnixBench shows that 2.50%

Motivation and Goal

Motivation

- An adversary occurs in kernel data modification through malicious code with kernel mode
- Enhancing kernel resilience at the kernel layer w/o any hardware and VMM features
  - Mitigate a kernel vulnerability attack with a memory corruption

The attack stages

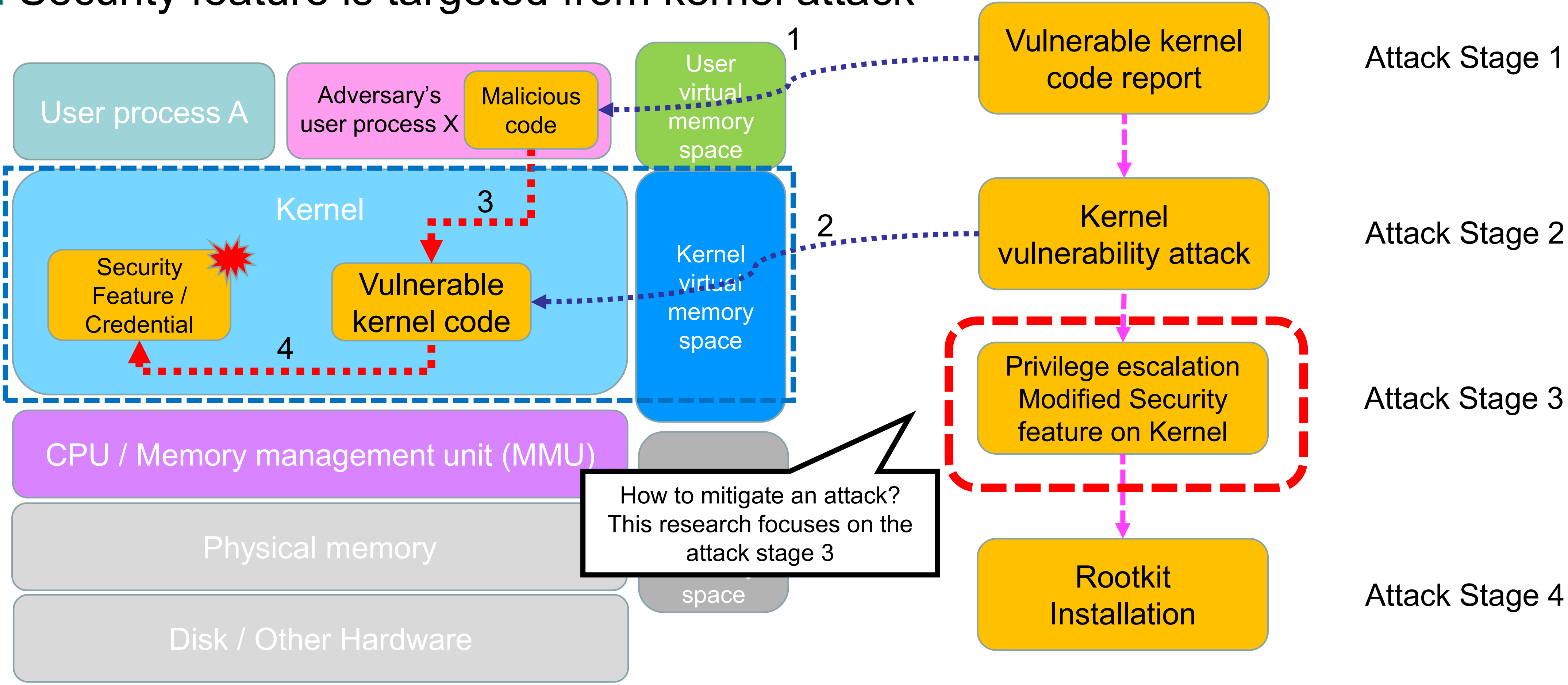


Goal

- To prevent illegal kernel data modification (i.e., credential information)
- Enhance the kernel security capability relies on a secure kernel mechanism

The detail of kernel vulnerability attack

- Security feature is targeted from kernel attack



User process A

Adversary's user process X

Malicious code

User virtual memory space

1

Kernel

3

Security Feature / Credential

Vulnerable kernel code

4

Kernel virtual memory space

2

CPU / Memory management unit (MMU)

Physical memory

Disk / Other Hardware

space

How to mitigate an attack?
This research focuses on the attack stage 3

Vulnerable kernel code report

Attack Stage 1

Kernel vulnerability attack

Attack Stage 2

Privilege escalation Modified Security feature on Kernel

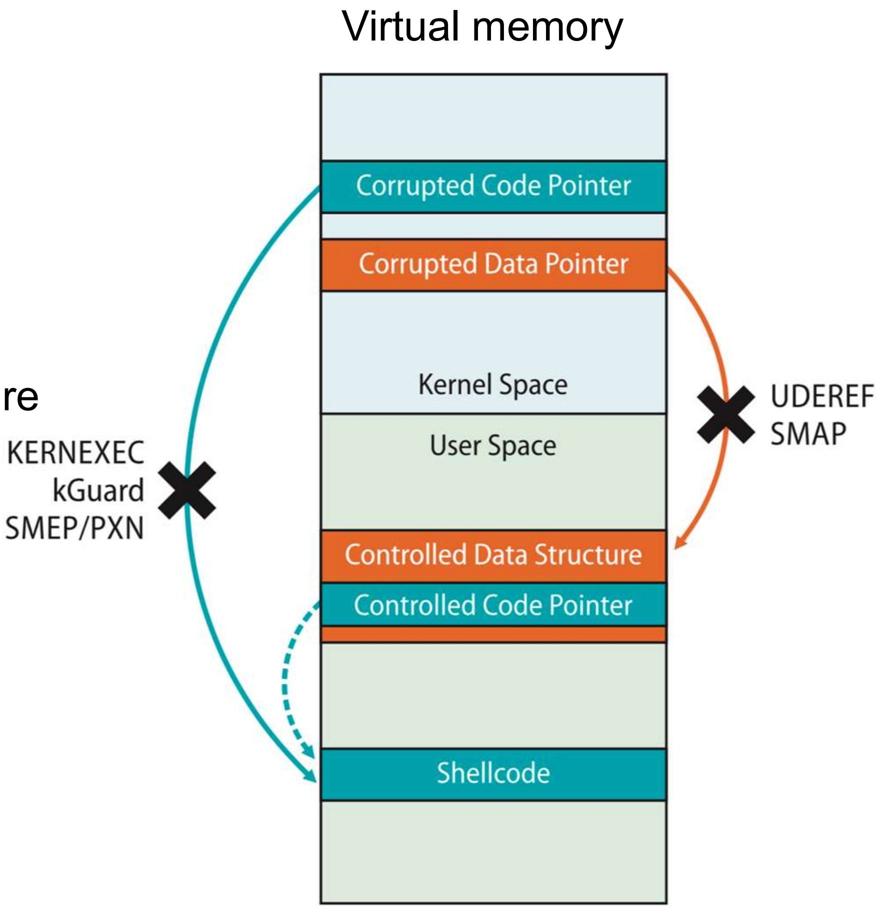
Attack Stage 3

Rootkit Installation

Attack Stage 4

Related work: Kernel vulnerability and countermeasures

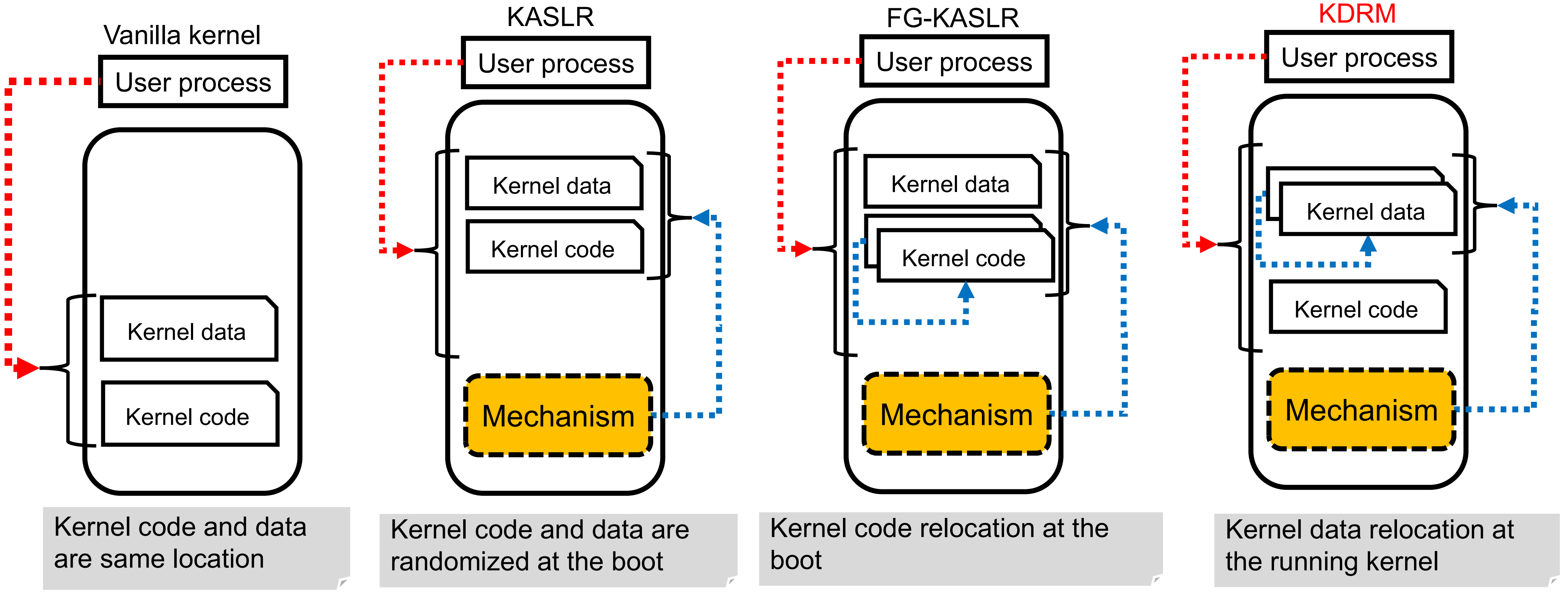
- Kernel vulnerability protections and attack[1]
  - KASLR: Kernel Address Layout Space Randomization
  - PaX: KERNEXEC, UDEREF
    - Memory fault has occurred on violation access
  - kGuard
    - Cross-platform compiler extension w/ out hardware feature
  - SMEP/SMAP/PXN/MPK
    - Supervisor Mode Execute Protection (SMEP)
    - Supervisor Mode Access Prevention (SMAP)
    - Privileged Execute-Never (PXN)
    - Memory protection key (MPK)：Protection key (Pkey)



[1]V Kemerlis, P, V., Polychronakis, M. and Kemerlis, D, A.: ret2dir: Rethinking Kernel Isolation. In: Proceedings of the 23rd USENIX Conference on Security Symposium, pp. 957-972, USENIX (2014).

Memory Randomization for Kernel Space

- Kernel hardening: it is difficult to identify the position of kernel code/data
  - The randomization work for kernel memory corruption or malicious invocation



Kernel code and data are same location

Kernel code and data are randomized at the boot

Kernel code relocation at the boot

Kernel data relocation at the running kernel

Problem, Threat model, and Contributions

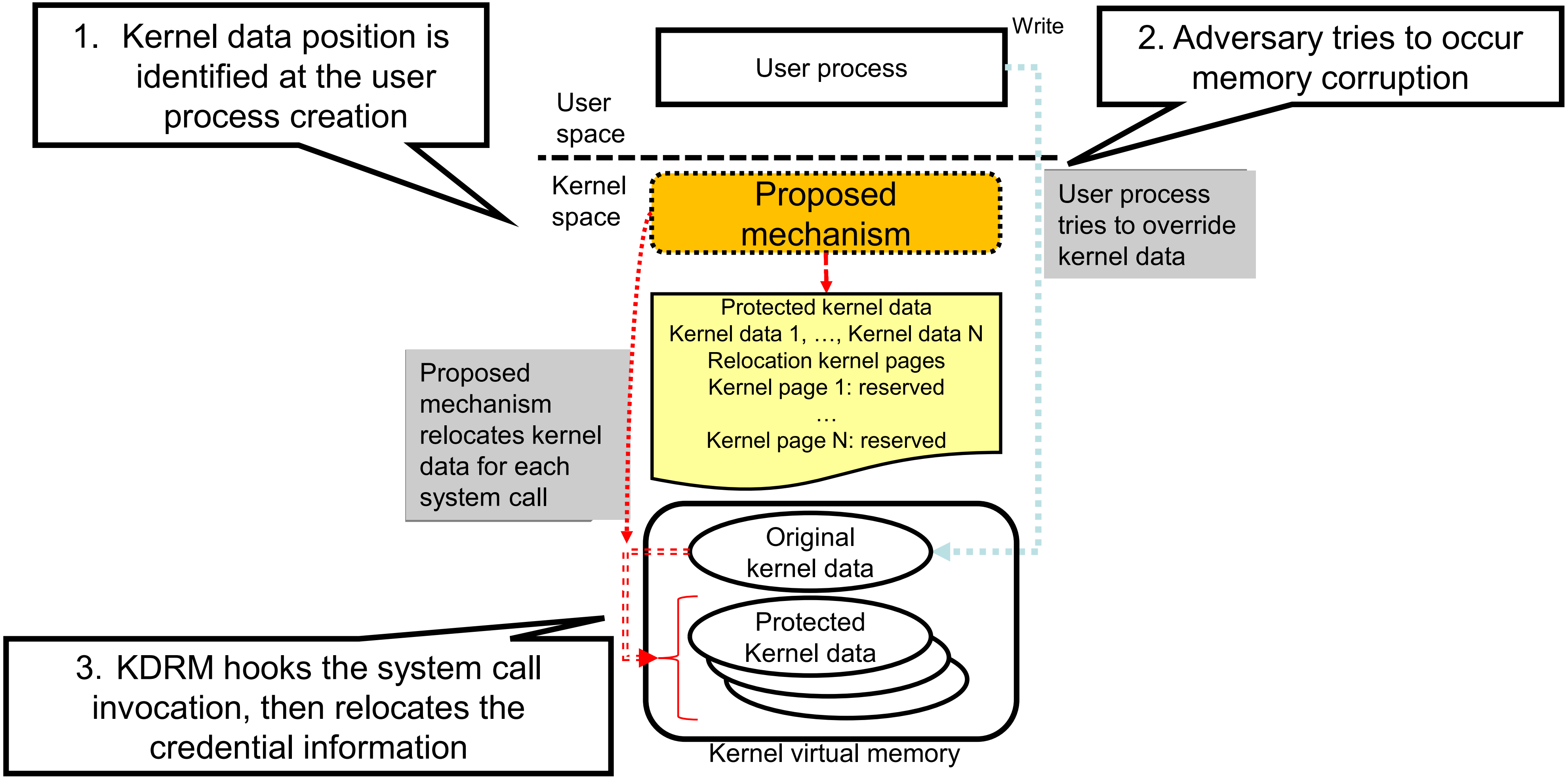
- Problem
  - The previous work randomized the position of kernel data at the kernel boot
    - After the kernel boot, kernel data is not randomized in its position on the kernel memory
    - If the adversary identifies the position, illegal modification is succeeded

  ⇒The approach mitigates the problem at a kernel layer

- Contribution
  - The purpose of the novel mechanism manages the relocation of kernel data (i.e., credential info)
    - KDRM changes the credential information of the user process at the running kernel
    - It protects credential information of user process from memory corruption
    - To avoid the miss behavior, KDRM works for system call invocations
  - The target attack is privilege escalation due to the implementation complexity of the Linux kernel

- Threat model
  - The adversary tries to invocate vulnerable kernel code that occurs memory corruption
  - Hardware is safe: BIOS, CPU, MMU, TLB

Approach of KDRM (Kernel data relocation mechanism)



1. Kernel data position is identified at the user process creation

User process

Write

User space

Kernel space

2. Adversary tries to occur memory corruption

User process tries to override kernel data

Proposed mechanism

Protected kernel data
Kernel data 1, …, Kernel data N
Relocation kernel pages
Kernel page 1: reserved
…
Kernel page N: reserved

Proposed mechanism relocates kernel data for each system call

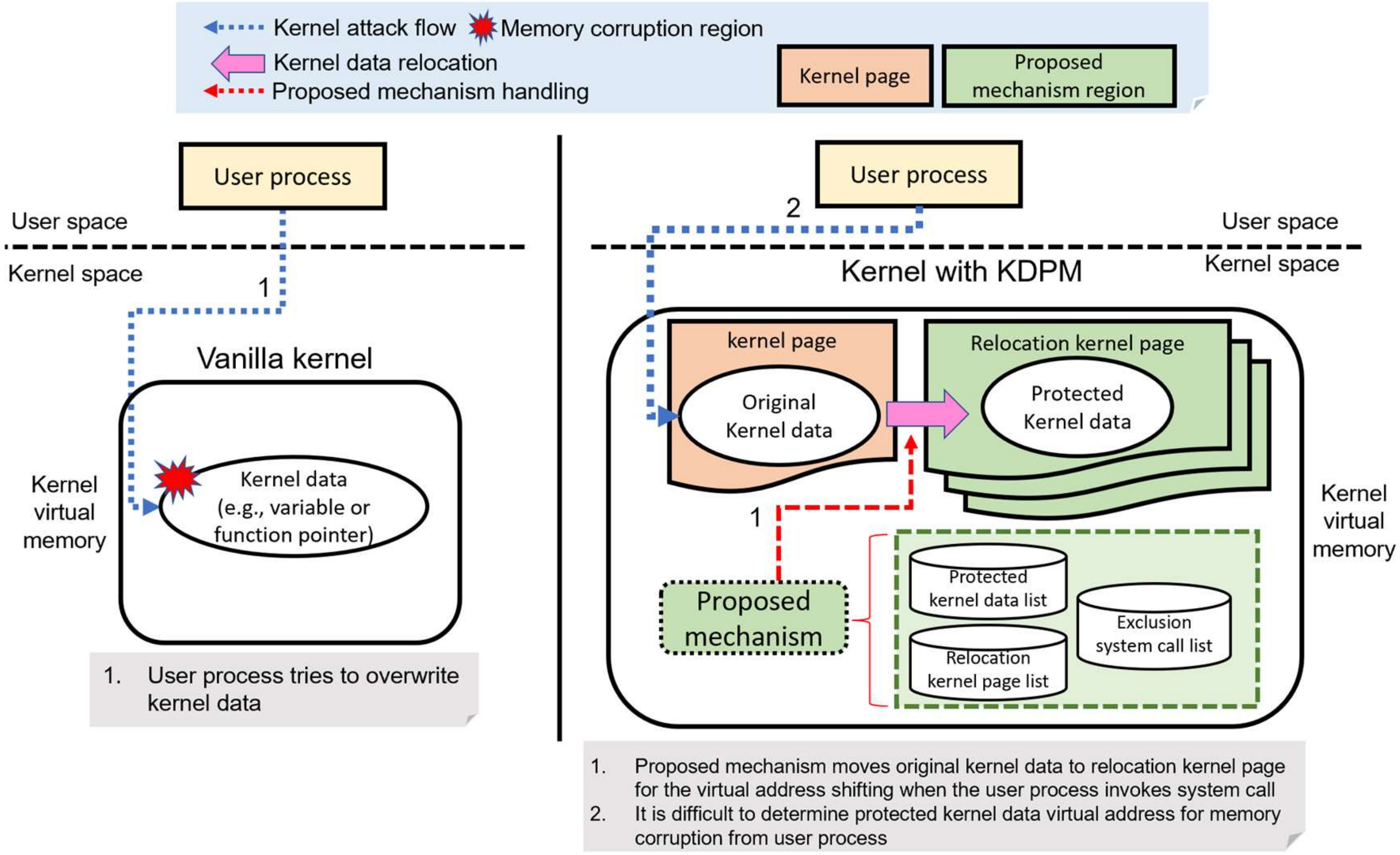
Original kernel data

Protected Kernel data

Kernel virtual memory

3. KDRM hooks the system call invocation, then relocates the credential information

Design of KDRM (Kernel data relocation mechanism)



1. User process tries to overwrite kernel data

1. Proposed mechanism moves original kernel data to relocation kernel page for the virtual address shifting when the user process invokes system call
2. It is difficult to determine protected kernel data virtual address for memory corruption from user process

Requirements

1. Memory corruption has occurred via system calls
2. Transparency for the user process
3. The relocation position is randomized

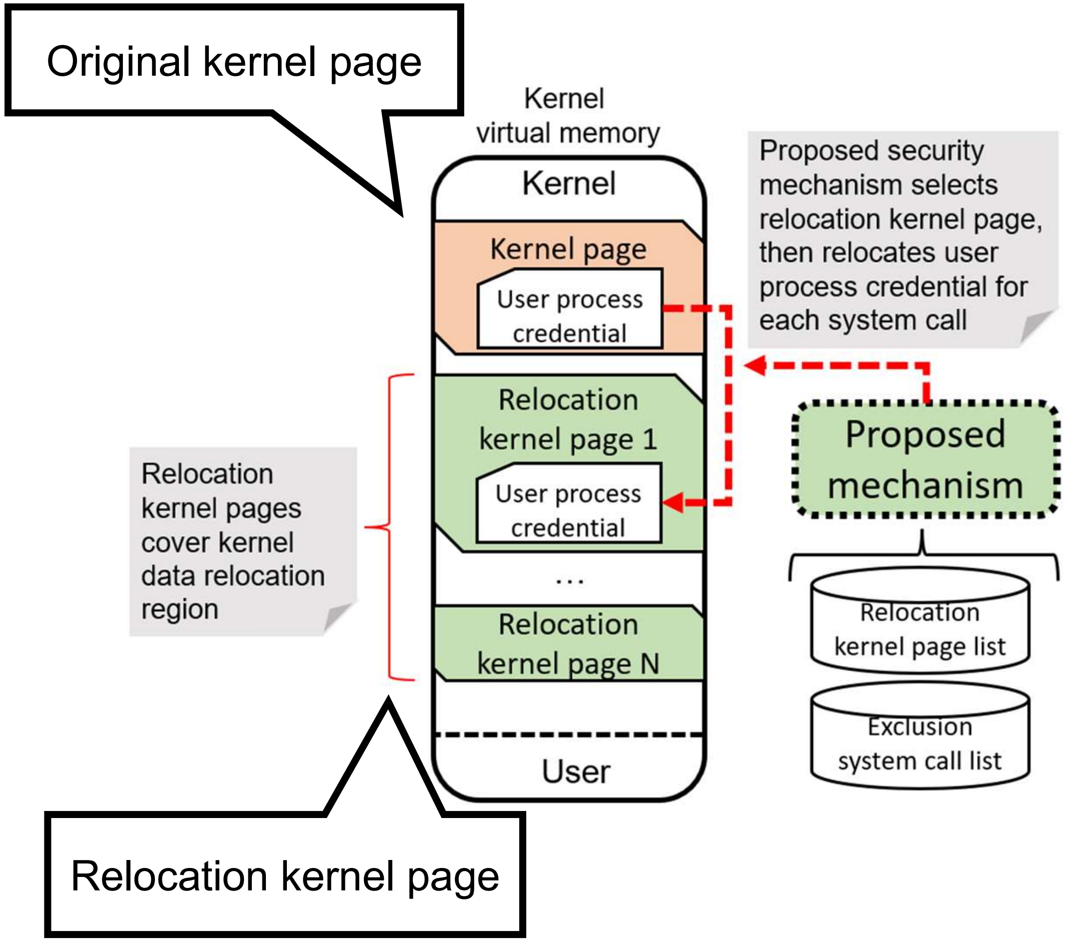
Design

1. The relocation mechanism is located in the kernel
2. Not affection for user process and kernel

Implementation

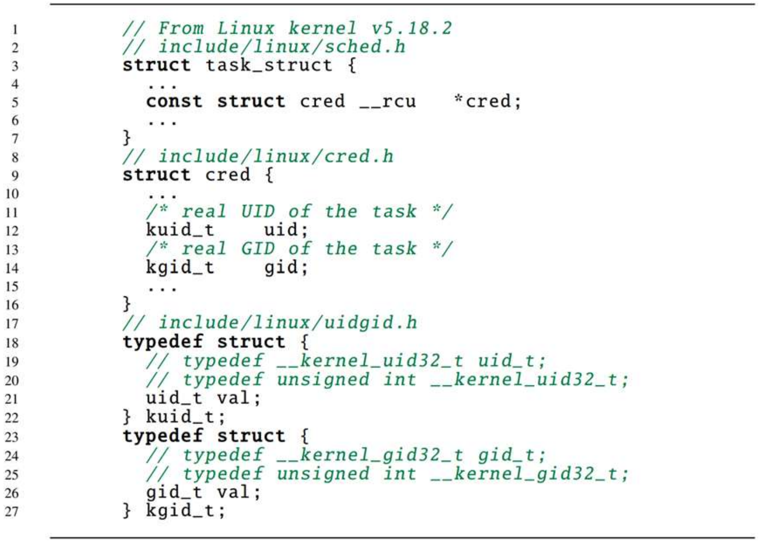
1. Target is credential information
2. Relocation kernel page is prepared

Implementation of KDRM (Kernel data relocation mechanism)



Proposed security mechanism selects relocation kernel page, then relocates user process credential for each system call

Relocation kernel pages cover kernel data relocation region

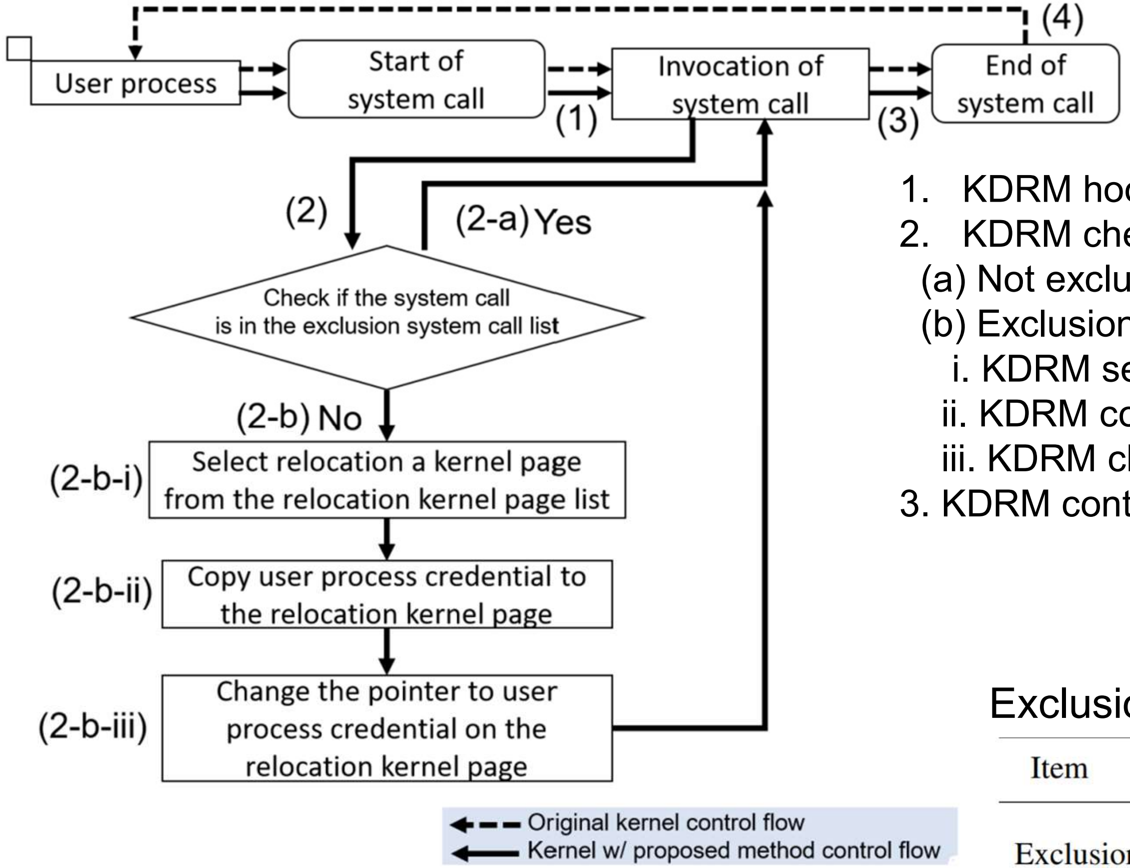


```
// From Linux kernel v5.18.2
// include/linux/sched.h
struct task_struct {
  ...
  const struct cred __rcu    *cred;
  ...
}
// include/linux/cred.h
struct cred {
  ...
  /* real UID of the task */
  kuid_t    uid;
  /* real GID of the task */
  kgid_t    gid;
  ...
}
// include/linux/uidgid.h
typedef struct {
  // typedef __kernel_uid32_t uid_t;
  // typedef unsigned int __kernel_uid32_t;
  uid_t val;
} kuid_t;
typedef struct {
  // typedef __kernel_gid32_t gid_t;
  // typedef unsigned int __kernel_gid32_t;
  gid_t val;
} kgid_t;
```

Implementation target

- Linux, x86_64
- Protected kernel data：User ID, Group ID
- Relocation kernel pages（4KB）
  - These are in task_sched for user process
- Explicit system call list
  - Credential management system call

Relocation Flow of KDRM



1. KDRM hooks system call invocations
2. KDRM checks the system call number
   (a) Not exclusion system call : relocation has not occurred
   (b) Exclusion system call: relocation has occurred
      i. KDRM selected the relocation kernel page
      ii. KDRM copies the credential information to it
      iii. KDRM changes the references to the relocation kernel page
3. KDRM continues the system call

Exclusion system call list

Item	Description
Exclusion system call list	execve, setuid, setgid, setreuid, setregid setresuid, setresgid, setfsuid, setfsgid

Evaluation

- Evaluations
  - 1. Privilege escalation attacks security assessment
    - Evaluation of kernel with KDRM can prevent privilege escalation attacks by introducing kernel vulnerabilities that can be used for memory corruption
  - 2. Performance evaluation in kernel operation
    - Benchmark software measures the effect of kernel feasibility and performance cost
  - 3. Attack difficulty assessment with kernel data relocation
    - The granularity of randomization of virtual addresses by the relocation of
    - kernel data using KDRM was compared with KASLR
- PoC attack code for evaluation: Kernel vulnerability
    - Return the value of the address of credential information, then write any data to it
- Environment : QEMU on the physical machine
  - CPU: Intel(R)Xeon(R) W-2295（3.00GHz，18コア，メモリ32GB），OS: Debian 11.3(x86 64)
  - Evaluation code：248行，PoC attack code：166行
  - Modified 12 source code files for Linux kernel 5.18.2

Evaluation1: Attack prevention of implementation

- Privilege escalation via introducing kernel vulnerability

Attack to credential information

```
// PoC code running, process id is 1676
1.   user $ ./a.out
2.   uid=1000(user) gid=1000(user) groups=1000(user)
3.   [*] sys_kvuln01 system call invocation
4.   uid virtual address: ffffffff820f0aef
5.   [*] sys_kvuln02 system call invocation
6.   Killed user process
```

2-6 lines：PoC code identifies the virtual address of the credential, then tries to overwrite it. However, the kernel occurred the page fault to kill the PoC code process

```
// Kernel log information
7.   [*] start user process
8.   [*] set kernel page of privilege at the user process creation
9.   [  363.704204] uid virtual address: ffffffff820f0aef
10.  [*] start system call invocation
11.  [  363.702116] sys_kvuln02 system call invocation
12.  [  363.702179] sysnum: 0x6a  (352)
13.  [  363.702204] PID: user process 1676
14.  [*] relocate kernel page of privilege
15.  [  363.704204] uid virtual address: ffffffff81099c50
```

8-15 lines：KDRM change the position of credential information before the attack is occurred

```
16.  [*] kernel code information
17.  // Kernel memory corruption
18.  [  363.704204] attack target virtual address: ffffffff820f0aef
19.  [  364.216821] page fault error code 2, virtual address: ffffffff820f0aef
     Page fault error code 2 (0b010)
20.  Page fault error code bits: from Linux v5.3.18 :
     arch/x86/include/asm/trap_pf.h
       a.   bit 0 == 0: no page found
       b.   bit 1 == 1: write access, X86_PF_WRITE
       c.   bit 0 == 0: kernel-mode access
21.  [*] finish system call invocation
22.  [*] finish user process
```

17-20 lines：KDRM catches the page fault for the previous virtual address of credential information with illegal write access

Evaluation2: Overhead measurement

- Performance evaluation results
  - 1. LMbench: 1 system call requires 0.0422 μs to 2.4341 μs overhead
  - 2. UnixBench: System call overhead, File copy, Pipe are effected. Score is 2.50% down

LMbench (us)

System call	Vanilla kernel	Implementation	Overhead
fork+/bin/sh	434.2899	446.8079	12.5180
fork+execve	101.2726	129.0260	27.7534
fork+exit	89.9990	94.8672	4.8682
open/close	1.1642	1.4920	0.3278
read	0.1177	0.1599	0.0422
write	0.0908	0.1359	0.0451
fstat	0.1484	0.1953	0.0468
stat	0.5265	0.6979	0.1714

UnixBench (score)

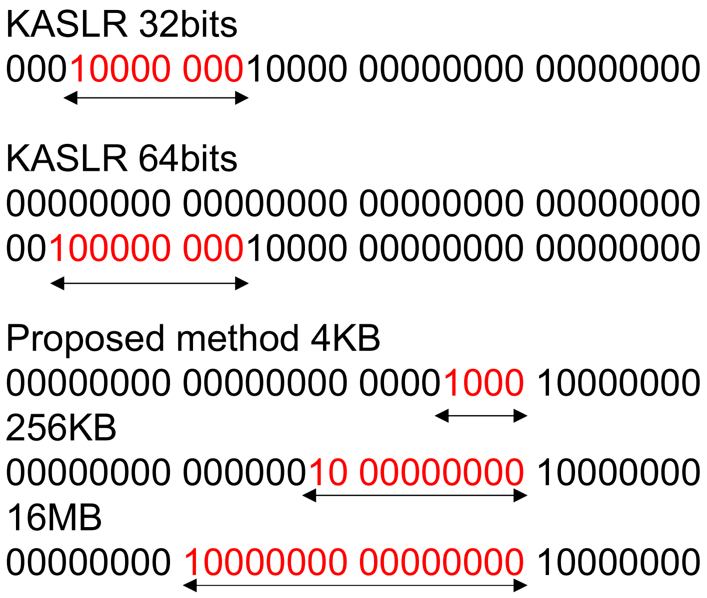
Instruction	Vanilla kernel	Implementation
Dhrystone 2	4450.50	4440.50 (0.22%)
Double-Precision Whetstone	1557.54	1552.92 (0.30%)
Execl Throughput	1193.23	1187.14 (0.52%)
File Copy 1024 bufsize	4122.08	3997.08 (3.03%)
File Copy 1024 bufsize	2790.40	2698.60 (3.29%)
File Copy 1024 bufsize	7401.80	7192.62 (2.82%)
Pipe Throughput	2109.68	2041.04 (3.25%)
Pipe-based Context Switching	806.02	785.34 (2.57%)
Process Creation	1019.10	1017.92 (0.12%)
Shell Scripts (1 concurrent)	2485.20	2456.13 (1.17%)
Shell Scripts (1 concurrent)	2298.00	2294.36 (0.16%)
System Call Overhead	1771.08	1620.68 (8.49%)
System Benchmarks Index Score	2195.16	2140.24 (2.50%)

Evaluaiton3: Attack difficulty assessment

- Randomization entropy：More than KASLR
- Attack estimation score：n bits for KASLR $1/2^{n-1}$, KDRM $1/2^{n}$
  - KDRM changed the position of credentials for each system call

Comparison of attack difficulty

Type	Entropy	Range	Align Size
Linux KASLR 32 bits	8 bits	512MB (29bits)	2MB (21bits)
Linux KASLR 64bits	9 bits	1GB (30bits)	2MB (21bits)
Proposed method	4 bits (1 page)	4 KB (12bits)	256 byte (8bits)
Proposed method	10 bits (64 pages)	256KB (18bits)	256 byte (8bits)
Proposed method	16 bits (4096 pages)	16 MB (24bits)	256 byte (8bits)



KASLR 32bits
00010000 00010000 00000000 00000000

KASLR 64bits
00000000 00000000 00000000 00000000
00100000 00010000 00000000 00000000

Proposed method 4KB
00000000 00000000 00001000 10000000
256KB
00000000 00000010 00000000 10000000
16MB
00000000 10000000 00000000 10000000

Discussion

- Evaluation results
  - Security capability for attack prevention of implementation

  ⇒KDRM mitigates privilege escalation to change the position of credential info
  - KDRM requires additional overhead for the invocation of system calls

  ⇒The reason for overhead relies on the replication of credential information
- Limitation
  - The attack complexity depends on the number of relocation kernel page
- Comparison of related works

Features	KASLR	KCoFI	KDRM
Protection target	Kernel data/code	Kernel code	**Kernel data**
Implementation	Memory place randomization	Verifying of kernel code invocation flow	Memory place relocation
Limitation	Only boot timing	Async kernel behavior	**Relocation number**

Conclusion and Future works

- KDRM presents the novel kernel security capability
  - It supports dynamically replacing credential information to change its virtual address
    - KDRM manages the relocation kernel page for the replacement target
  - It mitigates privilege escalation attacks via memory corruption
    - KDPM creates partially safe kernel data from directory illegal kernel data modification

  - Evaluation result
    - KDRM indicates the mitigation of privilege escalation via memory corruption
    - Overhead: 0.0422 μs to 2.4341 μs overhead, and 2.5% performance overhead
    - Attack difficulty assessment is compared with KASLR

- Future works
  - The consideration of the combination of other works, and evaluation of actual kernel vulnerability
  - Design portability for other OS and Architecture support

نشكركم جزيل الشكر على انتباهكم

Thank You-Merci-Gracias

kuzuno@port.kobe-u.ac.jp