KDRM: Kernel Data Relocation Mechanism to Mitigate Privilege Escalation Attack

Kuzuno, Hiroki

Yamauchi, Toshihiro

(Citation)
Network and System Security:61-76

(Issue Date)
2023-08-07

(Resource Type)
conference paper

(Version)
Accepted Manuscript

KDRM: Kernel Data Relocation Mechanism to Mitigate Privilege Escalation Attack

Hiroki Kuzuno [1] and Toshihiro Yamauchi [2]in

[1] Graduate School of Engineering, Kobe University, Japan
[2] Faculty of Environmental, Life, Natural Science and Technology, Okayama University, Japan
kuzuno@port.kobe-u.ac.jp, yamauchi@okayama-u.ac.jp

Abstract. A privilege escalation attack by memory corruption based on kernel vulnerability has been reported as a security threat to operating systems. Kernel address layout randomization (KASLR) randomizes kernel code and data placement on the kernel memory section for attack mitigation. However, a privilege escalation attack will succeed because the kernel data of privilege information is identified during a user process execution in a running kernel. In this paper, we propose a kernel data relocation mechanism (KDRM) that dynamically relocates privilege information in the running kernel to mitigate privilege escalation attacks using memory corruption. The KDRM provides multiple relocation-only pages in the kernel. The KDRM selects one of the relocation-only pages and moves the privilege information to the relocation-only pages when the system call is invoked. This allows the virtual address of the privilege information to change by dynamically relocating for a user process. The evaluation results confirmed that privilege escalation attacks by user processes on Linux could be prevented with KDRM. As a performance evaluation, we showed that the overhead of issuing a system call was up to 149.67%, and the impact on the kernel performance score was 2.50%, indicating that the impact on the running kernel can be negligible.

1 Introduction

Memory corruption countermeasures in the operating system (OS) kernel are paramount. In particular, privilege escalation attack and security-feature disabling attacks use memory corruption of kernel data [3,7].

In this regard, kernel address space layout randomization (KASLR) randomly places kernel code and data on the kernel memory at kernel startup as a countermeasure against memory corruption attacks. KASLR makes it challenging to identify the virtual address of the kernel data to be attacked and reduces the possibility of kernel data tampering due to memory corruption. However, the virtual address of kernel data (e.g., privilege information) on the kernel memory is fixed in the running kernel, which poses the following problem.

Privilege information illegal modification
Assume that an attacking user process has identified the virtual address of the privilege information to be attacked in a running kernel [8].

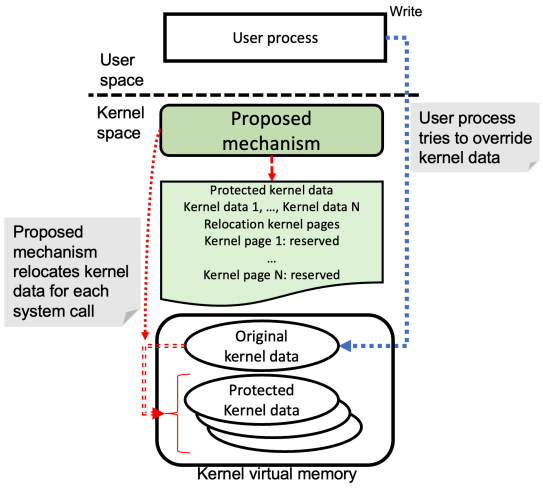

Fig. 1: Overview of the KDRM

The privilege information can be tampered with by an attack exploiting the kernel vulnerability. Therefore, an attacker can conduct a privilege escalation attack.

This paper proposes a kernel data relocation mechanism (KDRM) that allows dynamic relocation of privilege information in the kernel memory to provide the kernel with resistance to attacks against privilege escalation attack. The KDRM can be applied to a running kernel with KASLR to improve the attack resistance of the kernel.

Figure 1 provides an overview of the proposed KDRM. The KDRM forces the user process to relocate privilege information when a system call is issued. The relocation of kernel data contains privilege information that makes it challenging to identify the virtual address of privilege information.

The KDRM allocates multiple relocation-only pages (4 KB) for each user process in the kernel memory and uses them to store privilege information to be protected. When a system call is issued, the privilege information to be protected is replicated to a randomly selected relocation page, and the reference to privilege information to be protected is changed to the replicated page. Then, the original page is temporary unmapped by KDRM for the tampering protection in the kernel memory. The virtual address of privilege information in the running kernel is dynamically changed by changing the page and protected by unmapping the page where the privilege information is stored each time a system call is issued.

In the proposed approach, the difficulty in identifying the virtual address of privilege information depends on its size to be relocated and the number of relocation-only pages. When the relocation target is 256 bytes (8 bits) and one relocation-only page (4 KB, 12 bits) is set as one page, the privilege information

Table 1: Types and effects of kernel vulnerability [1]

	Item	Description
Type	Missing pointer check	Lack of pointer variable verification
	Missing permission check	Lack of permission verification
	Buffer overflow	Overwriting of the stack or heap space
	Uninitialized data	Lack of initialization at variable creation
	Null deference	Access to Null variable
	Divide by zero	Zero dividing calculation
	Infinite loop	Occurrence of the infinite loop process
	Data race / deadlock	Occurrence of race condition or deadlock
	Memory mismanagement	Inconsistent allocation of memory allocation and free
	Miscellaneous	Other wrong implementations
Effect	Memory corruption	Modification of kernel data
	Policy violation	Miss implementation of access control decision
	Denial of Service	Forcing kernel to stop running
	OS information leakage	Information leakage from uninitialized data variables

can be protected from brute force attacks in the range of 4bits (see section 6.7 for details).

Suppose that a user process performing a privilege escalation attack executes vulnerable kernel code that can be used in an arbitrary memory corruption attack. In that case, an attempt to tamper with privilege information can occur. However, the KDRM makes it challenging to locate the exact virtual address of privilege information. Therefore, tampering with privilege information occurs the page fault. In particular, tampering fails, and privilege escalation attacks are prevented.

The research contributions in this paper are as follows:

1. To mitigate memory corruption attacks, we designed and implemented a security mechanism that enables the dynamic relocation of privilege information when a system call is issued. We implemented the KDRM on Linux, which is resistant to privilege escalation attacks.
2. We confirmed that the KDRM could prevent privilege escalation against user processes that attempt privilege escalation attacks. In this regard, the impact of the KDRM on user process and kernel operation was evaluated. The results showed that the overhead to the kernel when issuing system calls ranged from 102.88% to 149.67%, and the impact on the kernel performance score was 2.50%.

2 Memory Corruption Vulnerability

Kernel vulnerabilities are mis-implementations that can be used to attack the kernel. Table 1 lists a classification of 10 types of kernel vulnerabilities and summarizes the effect of four types of attacks using kernel vulnerabilities [1].

The KDRM is a countermeasure against memory corruption attacks that exploit kernel vulnerabilities related to pointers and variables. A memory corruption attack is an attack that attempts to write an arbitrary virtual address in

```
1         // From Linux kernel v5.18.2
2         // include/linux/sched.h
3         struct task_struct {
4           ...
5           const struct cred __rcu    *cred;
6           ...
7         }
8         // include/linux/cred.h
9         struct cred {
10          ...
11          /* real UID of the task */
12          kuid_t   uid;
13          /* real GID of the task */
14          kgid_t   gid;
15          ...
16        }
17        // include/linux/uidgid.h
18        typedef struct {
19          // typedef __kernel_uid32_t  uid_t;
20          // typedef unsigned int  __kernel_uid32_t;
21          uid_t val;
22        } kuid_t;
23        typedef struct {
24          // typedef __kernel_gid32_t  gid_t;
25          // typedef unsigned int  __kernel_gid32_t;
26          gid_t val;
27        } kgid_t;
```

Fig. 2: Structures related to user ID in Linux[4]

the kernel memory. If the memory area to be attacked is rewritable, the attack target is overwritten by memory corruption.

Privilege Escalation Attack: For privilege escalation attacks, a kernel vulnerability that takes privilege management omissions to forcibly call kernel code that performs privilege modification operations [14,15,16] and memory corruption attacks has been reported [17].

The attacker attempts a privilege escalation attack that targets kernel data concerning user process privilege information, which is placed in the kernel memory. As a precondition for a successful privilege escalation attack, the virtual address of the kernel data that stores privilege information must be correctly specified as the attack target. After that, an attacker attempts to tamper with the kernel data that stores privilege information. Furthermore, the attacker can change the user ID of a user process to an administrator user in the case of a memory corruption attack.

Privilege Information: The attack target was kernel data related to privilege information. Figure 2 illustrates the structure definition of user ID in Linux. The `task_struct` structure in Linux manages user processes and stores privilege information. Line 5 shows that the privilege information is stored in the `cred` structure that manages the user process. The user ID is stored in the variable `uid` of the `kuid_t` structure on line 12, which is included in the `cred` structure on lines 9 through 16. The `kuid_t` structure has variable `val` of `uid_t` on lines

18 through 22. In a privilege escalation attack, the variable `val` of the `uid` of the `kuid_t` structure is rewritten to the user ID (0) of the `root`.

3 Threat Model

Attack Target Environment: The threat model in this study considered an attacker attempting a memory corruption via a kernel vulnerability. The attack target environment, assumed as the threat model, is summarized as follows:

- Attacker: Runs a user process with general user privileges. In the user process, the attacker executes the attack code, calls vulnerable kernel code and attempts a memory corruption attack.
- Kernel: Contains kernel vulnerabilities that can be used for memory corruption attacks, allowing user processes to call vulnerable kernel code. No security mechanisms other than access control functions are applied to user processes.
- Kernel Vulnerability: Specify an arbitrary virtual address of kernel data and achieve a memory corruption attack. Receive the virtual address and overwritten data of the attack target from the user process and tamper the kernel data of the attack target.
- Attack target: The attack target is kernel data placed on the kernel memory. The attack target stores user process privilege information.

Attack Scenario: In the assumed attack scenario, the attacker attempts a memory corruption attack against the kernel. In particular, the attacker executes an arbitrary user process as a normal user. The user process invokes vulnerable kernel code to perform memory corruption attacks that can overwrite the kernel data using arbitrary data.

For example, in a privilege escalation attack, an attacker identifies the position of the kernel data containing the privilege information of a user process. Subsequently, the attacker overwrites the privilege information of the user ID to administrative privilege.

4 The Design of The Approach

4.1 Requirement

The KDRM dynamically relocates protected kernel data during kernel operation. The design aims to satisfy the following requirements:

RQ1: The assumed attack is privilege information modification by a memory corruption attack through a kernel vulnerability during the execution of a system call.

RQ2: Relocation control of protected kernel data on the kernel is transparent to user processes.

RQ3: To make it difficult to identify the relocation position of privilege information.

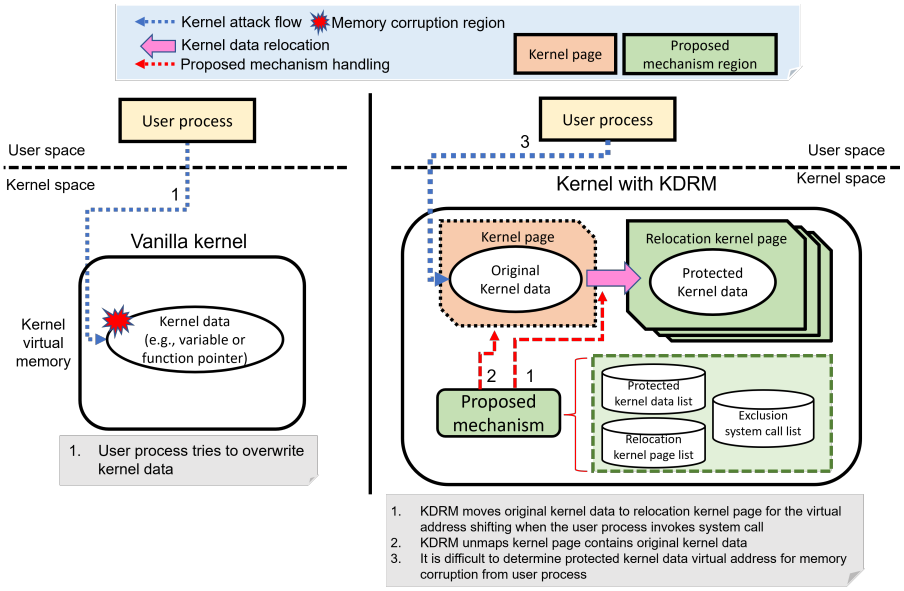

Fig. 3: Design overview of the KDRM

4.2 Concept

The design concepts of the KDRM are defined as follows:

- Concept 1: The protected kernel data relocation handling its performed in the kernel to mitigate attacks from user processes and make it challenging to detect countermeasures.
- Concept 2: The KDRM is designed to mitigate attacks on the protected kernel data relocation handling so that user processes and kernel operations are unaffected.

4.3 Protected Kernel Data Relocation Challenge

The design outline of the KDRM is shown in Figure 3. Based on the design concepts of the KDRM and to satisfy the requirements, multiple relocation-only pages are provided as relocation destinations of kernel data to be protected in the kernel on the kernel memory. In addition, a list of kernel data to be protected, a list of relocation-only pages, and a list of system calls that are excluded from relocation handling by the KDRM.

Protection Kernel Data: In the KDRM, the protected kernel data to be relocated on the kernel memory. The protected kernel data is privilege information that is created at the time of user process creation.

Relocation-only Page: In the KDRM, the relocation-only page is kernel page to which the protected kernel data is relocated. The KDRM provides multiple relocation-only pages on the kernel memory.

Table 2: Protected kernel data and Exclusion system call list

(a) Kernel data to be protected in the KDRM implementation

Item	Description
Protected kernel data	User ID (e.g., uid, euid, fsuid, and suid) Group ID (e.g., gid, egid, fsid, and sgid)

(b) A system call that performs authorization operation that exempts the realization method

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

Relocation Handling: The relocation handling of the protected kernel data in the KDRM is performed before and after the execution of the system call.

- Before system call execution: Protected kernel data is relocated to a relocation-only page and temporary unmapped the original kernel data from kernel memory.
- After executing the system call: The protected kernel data on the relocation-only page is moved to the original kernel data location.

In the KDRM, the relocation destination of protected kernel data is selected randomly from a list of relocation-only pages before executing the system call. The virtual address of the protected kernel data after relocation changes within a specific range, making it difficult to specify the virtual address.

5 Implementation

5.1 Implementation Overview

Linux on the x86_64 CPU architecture was assumed to be the environment for implementing the scheme. An overview of the implementation, the KDRM creates a new page and placed for each user process to make the privilege information of the kernel data to be protected. While processing of the implementation method, the page storing the privilege information is replicated to a randomly selected relocation-only page before the execution of the system call, and the virtual address of the privilege information is changed.

5.2 Protected Kernel Data

In the implementation method, a kernel page (4 KB) is created to store the protected kernel data as the privilege information of the user process.

Table 2a lists the privilege information of the protection target. During the operation period of the user process, each kernel page (4 KB) is subject to relocation handling.

5.3 Relocation kernel page

In the implementation, a certain number of relocation-only pages are allocated at kernel startup to reduce the load during user process creation. The `alloc_pages` function is used for allocating relocation-only pages. Multiple relocation-only pages (4 KB) (e.g., 10) are allocated when a user process was created. In addition, the `remove_pagetable` function is used for the unmapping original kernel page from the kernel page table that is the variable `pgd` of `current`.

A specific range of virtual addresses in the kernel memory can be used as relocation destinations for privilege information by allocating multiple relocation-only pages to each user process. In addition, the relocation-only page can be randomly selected, making it difficult to identify the virtual address of the relocation destination.

5.4 Relocation Handling

Relocation control of kernel data that stores privilege information is performed by using a list of relocation-only pages and a list of exempted system calls as follows:

1. Hooks system calls invocations by user processes.
2. Determine if the system call number is included in the list of exempted system calls.
   (a) For exempt system calls: privilege information is not relocated.
   (b) For other than exempt system calls: privilege information is relocated.
      i. Randomly selects a relocation-only page from the list of relocation-only pages as the relocation destination for privilege information.
      ii. Duplicate the kernel data storing the privilege information to the relocation-only page by page.
      iii. Change the reference from the privilege information in the kernel to the replication destination.
      iv. Unmap the privilege information of the original kernel page from the kernel page table.
3. Continue execution of the system call.
4. Terminate system call.

KDRM restores the privilege information of the original kernel page to the kernel page table after the termination of other than exempt system calls.

Page Fault Handling: An attempt to illegally overwrite kernel data that contain privilege. The page fault handler's `handle_page_fault` function catches a privilege escalation attack. The Linux kernel can know the referenced virtual address at the page fault. The implementation method compares the virtual address of the privilege information before relocation handling. As the page fault occurs, a SIGKILL is sent to the target user process with the function `force_sig_info` when considered an illegal write.

Protected Kernel Data Relocation Exemptions: Depending on the type of kernel data to be protected, reference or write failures to kernel data due to relocation in the kernel might affect the kernel and user process operations.

In particular, the KDRM allows the user to specify in advance which system calls are exempted from relocation for each protected kernel data to avoid affecting the kernel and user processes. Moreover, KDRM uses these system calls to determine whether the relocation handling is applicable. The kernel data to be protected is not relocated when the specified exempted system call is executed.

In the implementation, the writing to privilege information may cause page faults; Thus, system calls that explicitly write to privilege information are managed as a list of exemptions in Table 2b, which summarizes the system calls that operate the privilege information.

6 Evaluation

6.1 Evaluation Purpose

We evaluated the kernel with KDRM to investigate the security capability, the overhead to kernel processing, and the attack difficulty by relocating kernel data. The evaluation contents are listed as follows:

1. Privilege escalation attacks security assessment
   We evaluated whether the 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
   We used benchmarking software to calculate the kernel performance score with KDRM.
3. Performance evaluation in issuing system calls
   Using benchmark software, we measured the overhead of relocating kernel data before and after issuing system calls on a kernel with KDRM.
4. 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 to evaluate the attack difficulty.

6.2 Evaluation Environment

The evaluation device was used for security evaluation and performance evaluations. The evaluation device was an Intel(R) Xeon(R) W-2295 (3.00 GHz, 18 cores, 32 GB memory) running Debian 11.3, Linux kernel 5.18.2. We implemented the KDRM in Linux kernel 5.18.2 with 248 lines of code for nine files. Furthermore, we added 32 lines of kernel vulnerabilities that can be used for memory corruption for security evaluation to three files and implemented the PoC code in 134 lines.

6.3 Kernel Vulnerability

The following system calls were introduced to evaluate the security capability of KDRM:

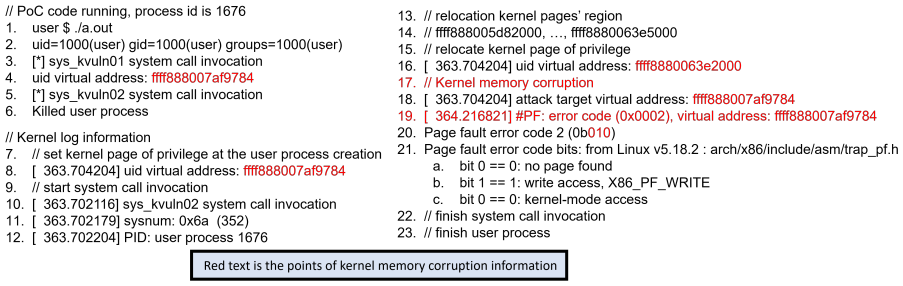

Fig. 4: Results of Preventing Privilege Elevation Attacks Using the KDRM.

– Original system call 1: Original system call 1 identifies the virtual address of the kernel data (e.g., the privilege information of the user process), then returns it to the user process.
– Original system call 2: Original system call 2 takes two arguments. The first argument is the virtual address, and the second is the overwritten data. Execution of the original system call 2 attempts to overwrite kernel data of the specified virtual address. A privilege escalation attack is possible if the first argument is the virtual address of the privilege information of the user process and the second argument is root ID (e.g., 0).

6.4 Privilege Escalation Attacks Security Assessment

As a security assessment, the attacking user process uses the original system call 1 to identify the virtual address of the privilege information and then attempts a privilege escalation attack using the original system call 2.

Figure 4 shows the attack prevention results of KDRM when a user process executes a privilege escalation attack.

In the attacking user process, line 2 displays the privilege information of the user process. The value of `uid` is 1,000, which confirms that the user is a normal user. In line 4, it calls the original system call 1 to specify the virtual address of the kernel data storing the privilege information.

In line 5, the user executes the original system call 2, a privilege escalation attack. In the kernel, line 8 shows the virtual address of the kernel data containing the privilege information. Lines 13 and 14 indicate the range of virtual addresses of the relocation-only page. In lines 15 and 16, KDRM moves the kernel data that stores the privilege information to the relocation-only page. The virtual address is changed before executing the original system call 2.

In line 18, an attempt is made to overwrite the virtual address specified by the original system call 2. A page fault with error number 2 is caught in line 19. This indicates a violation of writing to the page for the virtual address. The writing target is the previous virtual address of privilege information.

Table 3: UnixBench compile performance of implementation

	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 256 bufsize	2790.40	2698.60 (3.29%)
File Copy 4096 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 (8 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%)

6.5 Overhead of Kernel Performance

To evaluate the performance of the kernel, UnixBench version 5.1.3 was run five times on the Linux kernel before and after KDRM was applied, and the performance score was calculated from the average values.

Table 3 lists the UnixBench performance score of each running kernel for numerical computation, file copy, process processing, and system calls. Higher score values indicate high performance. From Table 3, the KDRM had most negligible impact on the score of 0.12% for Process Creation and the most significant impact on the score of 8.49% for System Call Overhead. The overall impact on the performance score was 2.50%.

6.6 Overhead of Kernel Processing

In KDRM, the privilege information is to be protected when a system call is performed. In the evaluation, we ran the benchmark software LMbench version 3.0-a9 10 times on a Linux kernel before and after applying KDRM. We calculated the overhead from the average value of the system call.

Table 4a lists the results of the performance evaluation. In LMbench, the number of system call invocations differs for each evaluation item: fork+/bin/sh is 54 times, fork+execve is 4 times, fork+exit is 2 times, open/close is 2 times, and the others are once.

6.7 Attack Difficulty Assessment by Kernel Data Relocation

A comparison of KDRM and the attack difficulty of Linux KASLR [13,5] is summarized in Table 4b. The randomization granularity of the virtual address was expressed in terms of entropy [19]. Moreover, 32 bits of Linux KASLR are randomized in 2 MB (21 bits) units, 512 MB (29 bits) has 8 bits of entropy, and 1 GB (30 bits) has 9 bits of entropy. In Table 4b, the relocation target is 256 bytes, 8 bits, and the relocation-only pages (4KB, 12 bits) are 1, 64, and 4096 pages with 4, 10, and 16 bits of entropy.

Table 4: Overhead and randomization entropy comparison

(a) Overhead of KDRM on the Linux kernel (μs)

System call	Vanilla kernel	Implementation	Overhead
fork+/bin/sh	434.2899	446.8079	12.5180 (102.88%)
fork+execve	101.2726	129.0260	27.7534 (127.40%)
fork+exit	89.9990	94.8672	4.8682 (105.41%)
open/close	1.1642	1.4920	0.3278 (128.16%)
read	0.1177	0.1599	0.0422 (135.85%)
write	0.0908	0.1359	0.0451 (149.67%)
fstat	0.1484	0.1953	0.0468 (131.60%)
stat	0.5265	0.6979	0.1714 (132.55%)

(b) The comparison of randomization entropy

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

The number of attack attempts required for successful memory corruption by a brute-force attack is $\frac{1}{2^{n-1}}$ times for n bits entropy if the virtual address is not changed during the attack attempts. If the virtual address can be randomized for each attack attempt, it is 2^n times [19]. Because Linux KASLR randomizes virtual addresses only at startup, the number of attack attempts, the result is $\frac{1}{2^n}$ times. Moreover, KDRM can randomize the virtual address of the kernel data at each system call of the user process; thus, number of attack attempts is 2^n times for n bits entropy.

7 Discussion

7.1 Evaluation Consideration

Evaluating the resistance to memory corruption attacks confirmed that the kernel with KDRM can mitigate privilege escalation attacks. When implementing KDRM, kernel data of privilege information is designated as a protection target, relocated, unmapped, and restored in the running kernel. Thus, making virtual address identification of the privilege information difficult.

The performance evaluation results show that the KDRM slightly affects the numerical calculations and process operations. However, the KDRM has a high overhead for processes requiring system calls, such as file copying. As a factor of overhead, we considered the processing time required to duplicate, unmap, and restore the protected kernel data after issuing the system call. The results confirmed that the stability of the kernel operation was not affected through performance evaluation.

7.2 Approach Consideration

Design and Implementation: The design of KDRM allows the relocation of protected kernel data at each system call issued to be transparent to user processes. We specified the user process privilege information stored in the kernel data at the time of user process creation because the privilege information is a target of privilege escalation attacks by memory corruption. To protect kernel

data other than privilege information, investigate the writing location for each kernel data and consider the related system calls.

System calls involving changes in privilege information are excluded from the application of KDRM to minimize the impact of KDRM on the kernel operation. In addition, if the kernel data to be protected exceeds the page size (4KB), or if many references in the kernel exist, the applicability of KDRM must be considered for performance impact.

Attack Difficulty: In the KDRM, the number of attack attempts against the protected kernel data is 2^n for n bits entropy. The attack cost is increased to make the memory corruption attack more challenging.

However, the n bit entropy increases or decreases depending on the size of the kernel data to be protected and the number of relocated pages. Thus, it is necessary to consider the difficulty of identifying virtual addresses and calculating the attack cost of memory corruption attacks depending on the type of kernel data.

7.3 Limitation of KDRM

The KDRM does not prevent vulnerable kernel code calls or illegal memory writes. CFI verifies the order of code calls and prevents unauthorized code calls. The Memory Protection Key (MPK) enables the CPU to limit writes on a page-by-page basis [10]. Therefore, we believe that combining CFI and MPK with KDRM can improve the attack resistance of the kernel.

7.4 Portability

The KDRM relies on managing the kernel memory per page to protect kernel data and the privilege information per user process. FreeBSD builds and manages the kernel memory using page tables and assigning privilege information to each user process [6]. We also believe that the KDRM can be implemented as a portability feature for FreeBSD.

8 Related Work

Running Kernel Protection: KASLR changes the kernel data and the virtual address of the kernel code to mitigate kernel memory corruption attacks [19]. Adelie proposes a method for extending KASLR to 64-bit and applying it to device drivers [18]. A method has also been proposed to apply KASLR to a guest OS from a virtual machine monitor [9].

Prevention Malicious Code Injection: As an attack prevention technique in the kernel, exclusive page frame ownership allows exclusive page allocation for the kernel and user processes [11]. KCoFI enables the kernel to apply control flow integrity (CFI), treating asynchronous processing as an exception and preventing incorrect code execution through code call order checking [2].

Table 5: Comparison of kernel data protection methods

Feature	KASLR [19]	KCoFI [2]	KDRM
Protection target	kernel code & data	kernel data	privilege information
Implementation	Memory layout randomization	Verifying control flow	Data relocation
Limitation	Kernel booting	Asynchronous	Relocation number

Kernel Attack Surface Reduction: As an attack surface minimization technique that removes attackable areas of the kernel, kRazor makes availability decisions on a per-kernel code basis during user process execution [12]. KASR places only the kernel code and data necessary for user process execution in memory space [20].

8.1 Comparison

Table 5 compares the proposed method with the previous studies [19,2].

In particular, KASLR changes the virtual address used for the kernel data access and kernel code calls at each kernel boot to make attacks more difficult [19]. In contrast, the virtual address locations of kernel code and kernel data do not change during kernel startup. The virtual address of kernel code or kernel data can be identified by a side-channel attack and used for the attack [8]. The KDRM performs kernel data relocation by using multiple relocation-only pages (4KB). The KDRM can be applied to a running kernel, and in combination with KASLR, it can improve its resistance to attacks.

KCoFI runs the kernel on its architecture and can verify the call order of asynchronous processing [2]. CFI is effective in preventing illegal code calls. In contrast, applying CFI to all kernel code calls in order increases the overhead. The KDRM features kernel data relocation and does not suppress attacks. Combined with CFI, the KDRM can prevent attacks when CFI is circumvented.

9 Conclusion

In this paper, we propose a KDRM that can relocate kernel data (e.g., privilege information) in the kernel memory to mitigate memory corruption attacks. The KDRM has multiple relocation-only pages, and privilege information is replicated to one of the randomly selected relocation-only pages. It ensures that allowing the placement of privilege information is changed dynamically and protected from privilege escalation. The KDRM can be used together with KASLR in the running kernel. In particular, identifying privilege information and privilege escalation attack is more challenging.

The evaluation results showed that privilege escalation attacks by user processes could be prevented. In the overhead evaluation, the kernel load when issuing system calls ranged from 102.88% to 149.67% with a kernel performance score of 2.50%. The attack difficulty evaluation of the kernel data relocation in KDRM indicates that this approach required more attack attempts than KASLR.

Acknowledgment

This work was partially supported by the Japan Society for the Promotion of Science (JSPS) KAKENHI Grant Number JP19H04109, JP22H03592, JP23K16882, and a contract of " Research and development on new generation cryptography for secure wireless communication services" among "Research and Development for Expansion of Radio Wave Resources (JPJ000254)". which was supported by the Ministry of Internal Affairs and Communications, Japan.

References

1. Chen, H., Mao, Y., Wang, X., Zhou, D., Zeldovich, N., Kaashoek, M.F.: Linux kernel vulnerabilities: State-of-the-art defenses and open problems. In: Proceedings of the Second Asia-Pacific Workshop on Systems. APSys '11, Association for Computing Machinery, NY, USA (2011). https://doi.org/10.1145/2103799.2103805
2. Criswell, J., Dautenhahn, N., Adve, V.: Kcofi: Complete control-flow integrity for commodity operating system kernels. In: Proceedings of 2014 IEEE Symposium on Security and Privacy. pp. 292–307 (2014). https://doi.org/10.1109/SP.2014.26
3. Database, E.: Nexus 5 android 5.0 - privilege escalation. https://www.exploit-db.com/exploits/35711/, accessed 21 May 2019
4. Foundation, L.: The linux kernel archives. https://www.kernel.org/, accessed 10 June 2022
5. Foundation, L.: Randomize the address of the kernel image (kaslr). https://www.kernelconfig.io/config_randomize_base, accessed 10 June 2022
6. FreeBSD: Freebsd architecture handbook. https://www.freebsd.org/doc/en_US.ISO8859-1/books/arch-handbook/, accessed 18 August 2019
7. grsecurity: super fun 2.6.30+/rhel5 2.6.18 local kernel exploit. https://grsecurity.net/~spender/exploits/exploit2.txt, accessed 21 May 2019
8. Gruss, D., Lipp, M., Schwarz, M., Fellner, R., Maurice, C., Mangard, S.: Kaslr is dead: Long live kaslr. In: Bodden, E., Payer, M., Athanasopoulos, E. (eds.) Engineering Secure Software and Systems. pp. 161–176. Springer International Publishing, Cham (2017)
9. Holmes, B., Waterman, J., Williams, D.: Kaslr in the age of microvms. In: Proceedings of the Seventeenth European Conference on Computer Systems. p. 149–165. EuroSys '22, Association for Computing Machinery, NY, USA (2022). https://doi.org/10.1145/3492321.3519578
10. Intel: Intel(r) 64 and ia-32 architectures software developer's manual. https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html, accessed 18 August 2021
11. Kemerlis, V.P., Polychronakis, M., Keromytis, A.D.: Ret2dir: Rethinking kernel isolation. In: Proceedings of the 23rd USENIX Conference on Security Symposium. p. 957–972. SEC'14, USENIX Association, USA (2014). https://doi.org/10.5555/2671225.2671286
12. Kurmus, A., Dechand, S., Kapitza, R.: Quantifiable run-time kernel attack surface reduction. In: Dietrich, S. (ed.) Detection of Intrusions and Malware, and Vulnerability Assessment. pp. 212–234. Springer International Publishing, Cham (2014)
13. LWN.net: Kernel address space layout randomization. https://lwn.net/Articles/569635/, accessed 12 May 2022

14. MITRE: Cve-2016-4997. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-4997, accessed 10 June 2019
15. MITRE: Cve-2016-9793. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-9793, accessed 10 June 2019
16. MITRE: Cve-2017-1000112. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-1000112, accessed 10 June 2019
17. MITRE: Cve-2017-16995. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-16995, accessed 10 June 2019
18. Nikolaev, R., Nadeem, H., Stone, C., Ravindran, B.: Adelie: Continuous address space layout re-randomization for linux drivers. In: Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems. p. 483–498. ASPLOS '22, Association for Computing Machinery, NY, USA (2022). https://doi.org/10.1145/3503222.3507779
19. Shacham, H., Page, M., Pfaff, B., Goh, E.J., Modadugu, N., Boneh, D.: On the effectiveness of address-space randomization. In: Proceedings of the 11th ACM Conference on Computer and Communications Security. p. 298–307. CCS '04, Association for Computing Machinery, NY, USA (2004). https://doi.org/10.1145/1030083.1030124
20. Zhang, Z., Cheng, Y., Nepal, S., Liu, D., Shen, Q., Rabhi, F.: Kasr: A reliable and practical approach to attack surface reduction of commodity os kernels. In: Bailey, M., Holz, T., Stamatogiannakis, M., Ioannidis, S. (eds.) Research in Attacks, Intrusions, and Defenses. pp. 691–710. Springer International Publishing, Cham (2018)