DirtyCred: Escalating Privilege in Linux Kernel

Zhenpeng Lin, Yuhang Wu, Xinyu Xing

Northwestern University

ACM CCS 2022
How Researchers Exploit Kernel Vulns

- Spatial/Temporal memory error
- Type confusion and memory overlap

(a) Type confusion between Type A and B
(b) Partial overlap between Type C and A
How Researchers Exploit Kernel Vulns

- Spatial/Temporal memory error
- Type confusion and memory overlap

(a) Type confusion between Type A and B
(b) Partial overlap between Type C and A
How Researchers Exploit Kernel Vulns

- Spatial/Temporal memory error
- Type confusion and memory overlap
- Leak kernel pointers
- Tamper kernel pointers

Partial overlap between Type C and A
How Researchers Exploit Kernel Vulns

- Spatial/Temporal memory error
- Type confusion and memory overlap
- Leak kernel pointers
- Tamper kernel pointers

Obtain Primitives

Bypass Mitigation
How Researchers Exploit Kernel Vulns

- Spatial/Temporal memory error
- Type confusion and memory overlap
- Leak kernel pointers
- Tamper kernel pointers
- Execute ROP in different forms[1]

[1] Joy of exploiting the kernel
How Researchers Exploit Kernel Vulns

- Spatial/Temporal memory error
- Type confusion and memory overlap
- Leak kernel pointers
- Tamper kernel pointers
- Execute ROP in different forms\footnote{Joy of exploiting the kernel}

\footnote{Joy of exploiting the kernel}
How Researchers Exploit Kernel Vulns

- Spatial/Temporal memory error
- Type confusion and memory overlap
- Leak kernel pointers
- Tamper kernel pointers
- Execute ROP in different forms

Used by 15/17 exploits in [2]

[1] Joy of exploiting the kernel
How DirtyCred Exploits Kernel Vulns

- Spatial/Temporal memory error
- Type confusion and overlap
- Leak kernel pointers
- Tamper kernel pointers
- Execute ROP
How DirtyCred Exploits Kernel Vulns

- Spatial/Temporal memory error
- Type confusion and memory overlap
- Leak kernel pointers
- Tamper kernel pointers
- Execute ROP

Obtain Primitives
How DirtyCred Exploits Kernel Vulns

- Spatial/Temporal memory error
- Type confusion and memory overlap
- Leak kernel pointers
- Tamper kernel pointers
- Execute ROP

Obtain Primitives
How DirtyCred Exploit Kernel Vulns

- Spatial/Temporal memory error
- Type confusion and memory overlap

Obtain Primitives

- Swap kernel credentials

Escalate Privilege
Kernel Credential

• Properties that carry privilege information in kernel
  • Defined in kernel documentation
  • Representation of privilege and capability
  • Two main types: task credentials and open file credentials

Source: https://www.kernel.org/doc/Documentation/security/credentials.txt
Task Credential

- **Struct cred** in Linux kernel’s implementation

```c
struct cred {
    atomic_t usage;
    #ifdef CONFIG_DEBUG_CREDENTIALS
        atomic_t subscribers; /* number of processes subscribed */
    void *put_addr;
    unsigned magic;
    #define CRED_MAGIC 0x43736564
    #define CRED_MAGIC_DEAD 0x44656144
    #endif

    kuid_t uid; /* real UID of the task */
    kgid_t gid; /* real GID of the task */
    kuid_t suid; /* saved UID of the task */
    kgid_t sgid; /* saved GID of the task */
    kuid_t euid; /* effective UID of the task */
    kgid_t egid; /* effective GID of the task */
    kuid_t fsuid; /* UID for VFS ops */
    kgid_t fsgid; /* GID for VFS ops */
};
```
Task Credential

- **Struct cred** in Linux kernel’s implementation
- Represents the *privilege* of kernel tasks

![Diagram showing Task Credential on kernel heap]
How Linux Kernel Uses Task Credential
How Linux Kernel Uses Task Credential
Open File Credential

• **Struct file** in Linux kernel’s implementation
Open File Credential

- Carries the information of opened files (e.g. mode, path, etc.)

```c
open("/tmp/a", O_RDWR)
open("/tmp/a", O_RDONLY)
open("/etc/passwd", O_RDONLY)
```
How Linux Kernel Uses Open File Credential

Write “0xdeadbeef” to the opened file

Kernel

User Space

Open File Credential

Read-only file
Read-write file
How Linux Kernel Uses Open File Credential

User Space

Kernel

Open File Credential

Write “0xdeadbeef” to the opened file

Error

Read-only file

Read-write file

Zhenpeng Lin
@Markak_

How Linux Kernel Uses Open File Credential

Write “0xdeadbeef” to the opened file

Error

Open File Credential

User Space

Kernel

/tmp/a
/etc/
passwd
freed

Read-only file

Read-write file
Attacking Task Credential

Kernel

- freed
- unprivileged
- privileged

User Space

Task Credential
Attacking Task Credential

Step 1. **Free** the *unprivileged* credential with the vulnerability
Attacking Task Credential

Step 1. **Free** the *unprivileged* credential with the vulnerability
Step 2. **Allocate** a privileged credential in the **freed** memory slot.
Attacking Task Credential

Result: Becoming a *privileged* user
Attacking Task Credential

Result: Becoming a *privileged* user

![Diagram showing the process of becoming a privileged user by attacking a task credential.]
Attacking Open File Credential

Kernel

/User Space

Write Syscall

Open File Credential

Read-only file

Read-write file
Attacking Open File Credential

Step 1. **Free** a *read-write* file *after* checks, but *before* writing to disk.
Attacking Open File Credential

Step 1. Free a *read-write* file *after* checks, but *before* writing to disk.
Attacking Open File Credential

Step 2. Allocate a *read-only* file in the *freed* memory slot
Attacking Open File Credential

Result: Writing content to read-only files
Challenges

1. How to free credentials.
2. How to allocate *privileged* credentials as *unprivileged* users. (attacking *task* credentials)
3. How to finish attack in a *small* time window. (attacking *open file* credentials)
Challenges

1. How to free credentials.

2. How to allocate privileged credentials as unprivileged users. (attacking task credentials)

3. How to finish attack in a small time window. (attacking open file credentials)
Challenge 1: Free Credentials Invalidly

- Both `cred` and `file` object are in dedicated caches.
- Most vulnerabilities happen in generic caches.
Challenges

1. How to free credentials.

2. How to allocate privileged credentials as unprivileged users. (attacking task credentials)

3. How to finish attack in a small time window. (attacking open file credentials)
Challenge 2: Allocating Privileged Task Credentials

• *Unprivileged* users come with *unprivileged* task credentials

• Waiting privileged users to allocate task credentials influences the success rate
Challenges

1. How to free credentials.

2. How to allocate privileged credentials as unprivileged users. (attacking task credentials)

3. How to finish attack in a small time window. (attacking open file credentials)
Challenge 3: Wining the race

- Kernel will examine the access permission before writing to the disk

![Diagram showing the process of kernel and user space interactions related to file permissions and writes.](image-url)
Challenge 3: Wining the race

- The swap of file object happens before permission check.
Challenge 3: Wining the race

- The swap of file object happens before permission check.
Challenge 3: Wining the race

- The swap of *file* object happens before *permission check*

![Diagram showing the process of file manipulation and permission checks.](image)

- Swapping
- Check perm
- Write to disk
- Error
- Write Syscall
Challenge 3: Wining the race

- The swap of file object happens after file write.

![Diagram showing the process of file write and the race condition.]

- Check perm
- Write to disk
- Swapping
- Write Syscall

User Space

Kernel

Open File Credential

/tmp/a  /tmp/a  freed

Read-only file

Read-write file
Challenge 3: Wining the race

- The swap of file object happens after `file write`.

![Diagram showing file system operations]

- Check perm

- Write to disk

- User Space:
  - Open File Credential
  - freed

- Kernel:
  - /tmp/a
  - /tmp/a
  - freed

- Read-only file
- Read-write file
Challenge 3: Wining the race

- The swap of file object happens after file write.
Challenge 3: Wining the race

- The swap of file object happens after file write.
Challenge 3: Wining the race

- The swap happens in between *permission check* and *file write*
Challenge 3: Wining the race

- The swap happens in between *permission check* and *file write*.
Challenge 3: Wining the race

- The swap happens in between *permission check* and *file write*
Challenge 3: Wining the race

- The swap must happen after *permission check* and before *file write*.
How We Address The Challenges

DirtyCred: Escalating Privilege in Linux Kernel

Zhengpeng Lin
Northwestern University

Yulang Wu
Northwestern University

Xinyu Yang
Northwestern University

ABSTRACT

The kernel vulnerability DirtyCred was revealed to be present in Linux, an open-source operating system, and was exploited to escalate privilege in Linux without requiring root access. This vulnerability is particularly dangerous because it can be exploited by any user who can gain access to the kernel. It is important to be aware of this vulnerability to protect against potential security threats.

1 INTRODUCTION

In this paper, we introduce the concept of a MilkyWay kernel vulnerability, which refers to a vulnerability that allows an attacker to escalate privilege in the Linux kernel without requiring root access. This vulnerability is particularly dangerous because it can be exploited by any user who can gain access to the kernel. It is important to be aware of this vulnerability to protect against potential security threats.

CCS CONCEPTS
- Security and privacy → Operating system security; Software security engineering;

KEYWORDS
- MilkyWay
- Kernel vulnerability
- Operating system security
- Software security engineering
Real-World Impact of DirtyCred

- **CVE-2021-4154**
  - Received rewards from Google’s KCTF
  - The exploit works across kernel v4.18 ~ v5.10
- **CVE-2022-2588**
  - Pwn2own exploitation
  - The exploit works across kernel v3.17 ~ v5.19
- **CVE-2022-20409**
  - Received rewards from Google’s KCTF and Android
  - The exploit works on both Android and generic Linux kernel
Defense Against DirtyCred

• Fundamental problem
  • Object isolation is based on type not privilege

• Solution
  • Isolate privileged credentials from unprivileged ones

• Where to isolate?
  • Virtual memory (privileged credentials will be vmalloc-ed)

All codes are available at https://github.com/markakd/DirtyCred
Summary

• A new exploitation concept — DirtyCred
• Principled approaches to different challenges
• A way to produce *Universal* kernel exploits
• Effective defense with negligible overhead

Zhenpeng Lin (@Markak_)

https://zplin.me

zplin@u.northwestern.edu

Logo comes from @sirdarckcat