Graham Dungan
April 3, 2025
UIN: 332001764

CSCE 413: Software Security
Self-Study: DirtyCow

What is the Vulnerability?

DirtyCow is a local privilege escalation exploit that uses a memory mapping vulnerability to allow users to modify kernel-protected files. DirtyCow existed for nearly nine years before being discovered in October of 2016, having been first introduced in 2007. This exploit is performed by abusing a race condition that exists when creating and modifying copy-on-write (CoW) pages while simultaneously disposing of them with the `madvise` system call. Due to this race condition, the computer mistakenly allows writes to be performed on read-only memory (instead of the private copies made), creating dirty pages that are eventually written to disk. Using this vulnerability, non-privileged users can write to privileged files to modify passwords, permissions, or the behavior of certain programs.

What was the Root Cause?

The root cause of this vulnerability was the race condition between the `madvise` system call and `write`. To reduce the cost of copying, the Linux operating system performs a copy-on-write (CoW) protocol whenever copying pages. Instead of allocating new memory and then copying data over, Linux will allow the user to read directly from the original page. If a write is made, the kernel will make a private copy of that page to apply the new written data.

The user can request a new mapping of certain pages via the `mmap` system call. The user can then use two threads to produce a DirtyCow attack- the first thread will constantly call `madvise` to inform the kernel that a certain memory region is not needed and that the system can reclaim the memory. The second thread will write to the same mapping in its own allocated memory region `/proc/self/mem` (to circumvent the `PROT_READ` protecting on the mapped page). Due to the race condition, the system will *think* the page is dropped due to `madvise` but will handle the `write` call before the page is actually remapped. This results in the write going directly to the original read-only page, instead of the private copy. Since the original page has been written to, and is now dirty, the kernel writes it to disk. This now allows a user to modify any file they can read, replacing files and escalating privileges.

What is the Extent of its Impact?

Phil Oester, a Linux security researcher, discovered DirtyCow in October of 2016. DirtyCow affected all Linux-based distributions and their former releases from the previous nine years leading up to the disclosure. Despite the relatively quick patch of DirtyCow across these operating systems, the failure to upgrade systems left many vulnerable devices across the internet. While there are few documented incidents of DirtyCow being used to explicitly gain root-level access on desktop computers (even though there were incidents), some of the most notable malware that resulted from this vulnerability affected Android devices. ZNIU was the first Android-based use of DirtyCow to perform privilege escalation on Android devices, obtaining root access and installing a backdoor. Over time, nearly 1,200 malicious Android apps were found to contain this exploit.

How to Patch it?

The kernel used a function named `get_user_pages` (also known as `gup`) to verify the status of pages before writing. However, due to a lack of checks, this function did not verify if the page was writable before performing writes. Following, the `madvise` function did not properly lock memory regions before it was used, leaving it exposed to the race condition, which did not prevent pages from being invalidated *while* another thread was preparing to write. Finally, as mentioned earlier, developers prevented the workaround of using `/proc/self/mem` to circumvent memory protections.

How Could it Have Been Prevented?

Ironically, Linus Torvalds admitted to attempting to fix this vulnerability eleven years ago, "This is an ancient bug that was actually attempted to be fixed once (badly) by me eleven years ago in commit 4ceb5db9757a ... but that was then undone due to problems on s390 by commit f33ea7f404e5...". While this vulnerability was somewhat known many years before its discovery, this exploit falls under the realm of wicked problems. There is ultimately an unknown number of security vulnerabilities existing within every operating system that require harsh experimentation, use, and bug testing to be found. In this instance, a further inspection of the operating system eleven years ago would have prevented this vulnerability, but saying "it just takes more scrutiny" undermines the difficulty of finding these near-invisible bugs in the first place.

References

1. Explaining Dirty Cow - Computerphile
2. Explaining Dirty COW local root exploit - CVE-2016-5195
3. Dirty COW (CVE-2016-5195)
4. dirtycow.github.io
5. Understanding and mitigating the Dirty Cow Vulnerability
6. index : kernel/git/torvalds/linux.git
7. First Android Malware Found Exploiting Dirty COW Linux Flaw to Gain Root Privileges
8. Dirty COW - Wikipedia
9. The Dirty Cow Linux bug: A silly name for a serious problem
10. ZNIU: First Android Malware to Exploit Dirty COW