2022 Security Summary Report of Live Detection of CVE20220847 Dirty Pipe

Prepared For

Report Issued: December 4, 2022

Report Prepared By Team B:
Katrina Siegfried
Xen Hasnat
Matthew Taylor
Aaliyah Sibug

TABLE OF CONTENTS

Introduction

Team B tested whether the *Dirty Pipe (CVE20220847)* exploit on the Linux 5.8.0 kernel could be detected in real-time. In this paper, TeamB explains what the dirty pipe exploit is, how to set up a testing environment to simulate the exploit, and how to use detection tools to detect the attack in real-time. The purpose of this study was to identify whether the *Dirty Pipe* exploit could be detected in real-time.

Security Implications

The security implications of this vulnerability are quite severe. Someone can arbitrarily overwrite any file on the system with the right commands and with only read permissions. In other words, if a user has read access over the file, they can also write to it. This also applies to read-only file systems, or otherwise protected files which the kernel would usually stop us from writing to; by exploiting the kernel vulnerability and circumventing the usual write methods, we also bypass these protections. It's important to note that the changes will not be permanent as the exploit is only writing to volatile memory in the page cache. The changes made through this exploit will remain in the cache until the kernel chooses to reclaim the memory allocated by the cache. Restarting the device, clearing the page cache manually, or the kernel reclaiming the memory will revert the file back to its original contents.

With root access, an attacker can do anything to the host machine. For example, the attacker can inject malware and delete or alter data. The ramifications of such an attack can be severe and depend on the nature of the target machine.

EXPLOIT DETAILS

Background

In 2016, two new functions were introduced to the Linux kernel that can allocate new pipe buffers. However, this commit missed the initialization of the flags variable - making it possible to create page cache references with arbitrary flags. This was not a big issue until anonymous pipe bugger merging was introduced in Linux 5.8 (PIPE_BIF_FLAG_CAN_MERGE) making it possible to overwrite data within page caches. Since the kernel is in full control of the page cache, no permission checking is done when writing to pipe.

Requirements

There are several requirements for this exploit; attacker must have read permissions, an writable offset must not be on a page boundary, write cannot cross a page boundary, and the file cannot be resized.

High-Level Process

The first step is to create a *pipe*. Then, the attacker fills the pipe with arbitrary data and sets the *PIPE_BUF_FLAG_CAN_MERGE* flag in all ring entries. Next, an attacker drains the pipe and leaves the `struct pipe_buffer` instances on the `struct pipe_inode_info ring`. Normally, this flag would be reset, however the bug used by this exploit leaves the value as 1. After that, the attacker splices data from the target file opened with `O_RDONLY` into the pipe from just before the target offset using the *splice() system call*. The last step is to write arbitrary data into the pipe, this data will overwrite the cached file page instead of creating a new anonymous `struct pipe_buffer` because `PIPE_BUF_FLAG_CAN_MERGE` is set.

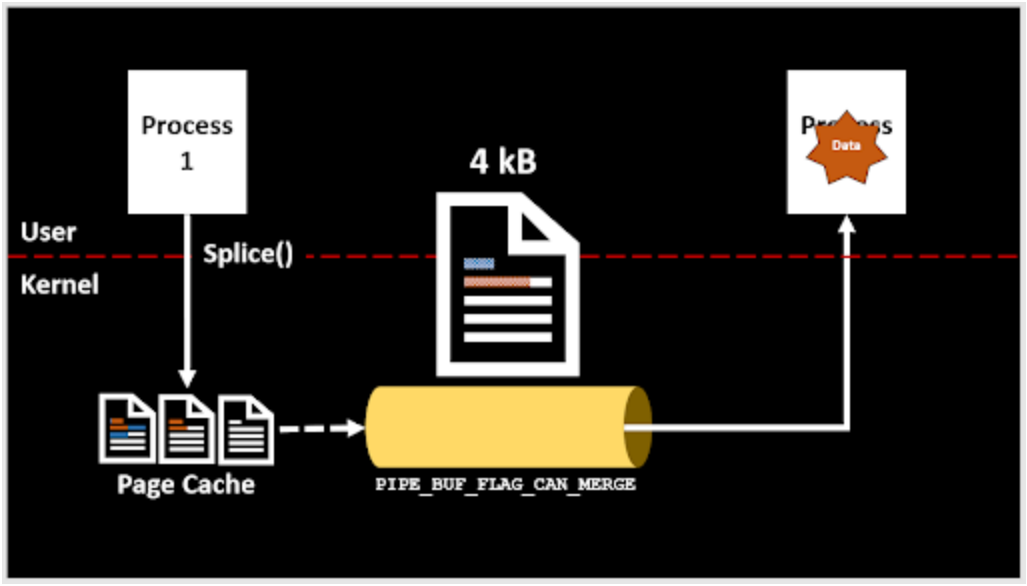

Figure 1: The dirtypipe exploit utilizes the PIPE_BUF_FLAG_CAN_MERGE_FLAG for the pipe buffer, which, when paired with a splice system call, allows overwriting of the page cache file contents, even in a read-only pipe with read-only permissions

Definitions

- *Pipe:* a pipe is a form of redirection (transfer of standard output to some other destination) that is used in Linux and other Unix-like operating system to send the output of one command/program/process to another command/program/process
- *Memory Page:* 4066-byte block of virtual memory with defined length (smallest data unit for memory management)
- *Page Cache:* a subsystem in the kernel that handles memory pages. When a file is read, data is put into page cache to avoid using disk. Data is also placed in the page cache when writing to a file before getting into storage. The “page cache” in the dirty pipe exploit becomes “dirty” because its altered from what is on the disk
- *Pipe Flags:* denote status and permission for data in pipe
- *PIPE_BUF_FLAG_CAN_MERGE:* this pipe defines whether the data buffer inside the pipe can be merged
- *System Call (syscalls)*: send requests to the kernel from userspace
- *Splice():* a syscall that can move data between file descriptors and pipes without userspace interaction

Scope

The items in scope are listed below.

- ➢ Research and information gathering of *Dirty Pipe (CVE20220847)* exploit
- ➢ Creation of a network environment to perform the *Dirty Pipe (CVE20220847)* exploit
- ➢ Execution of *Dirty Pipe (CVE20220847)* exploit on an Ubuntu virtual machine with Linux kernel 5.8.0
- ➢ Exploitation of strategies for live detection scanning of *Dirty Pipe (CVE20220847)* exploit.

TESTING METHODOLOGY

CSCI 5403 / CYBR 5300 Team B's testing methodology was split into three phases: *Reconnaissance and Information Gathering*, *Execution of Vulnerabilities, and Live Detection*. During reconnaissance, we gathered information about Linux kernel exploit CVE20220847, also known as dirtypipe. Team B researched the nature of the dirty pipe exploit and mapped out a network of machines to replicate the exploit and explore potential strategies for live detection scanning. The following is a graphical representation of our network setup.

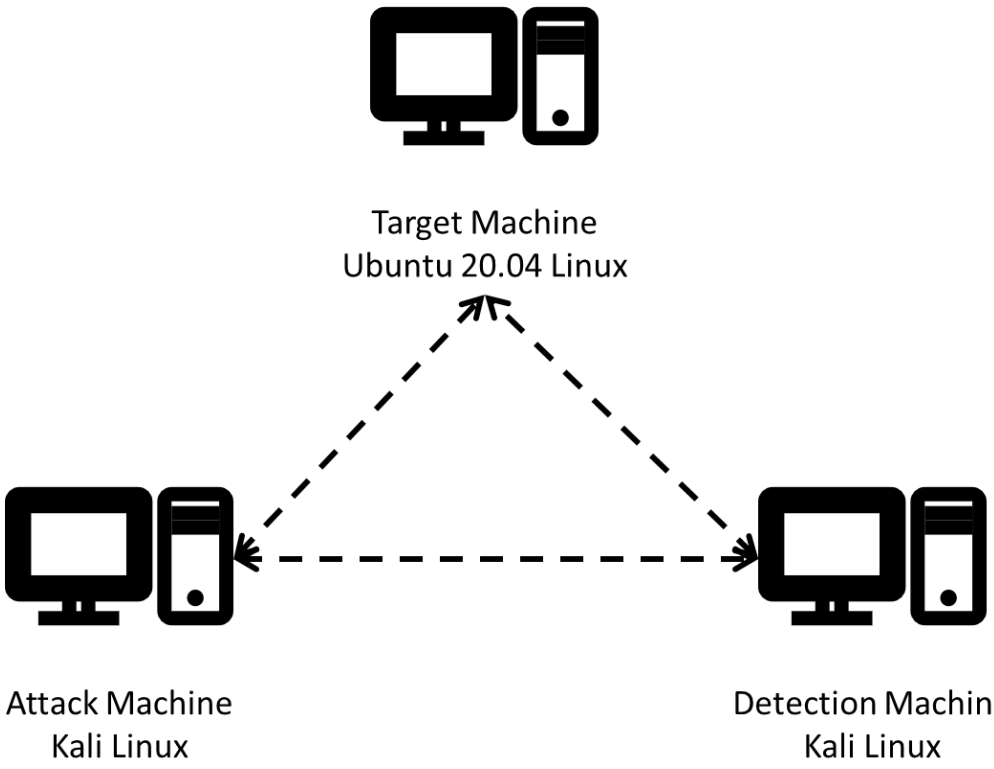

Figure 2: A diagram of the network configuration used in the exploitation and live detection.

Reconnaissance and Information Gathering

The Ubuntu 20.04 LTS with the linux 5.8.0 kernel was selected as a target because it is both a common distribution configuration and has been identified as having been vulnerable to the CVE20220847 dirtypipe exploit. The Kali Linux machines were selected for the attack and detection machines because they are pre-equipped with the tools and programs necessary in

exploit execution and detection. The three machines were created as virtual machines on a single host, with network configurations for host-only. All machines were configured to deny promiscuous mode with the exception of the detection Kali Linux machine which was configured to ‘Allow All’.

Execution of Vulnerabilities

Team B then created a demo program written in C which was written using information from several different sources, primarily the original demo code from the exploit’s discoverer Max Kellermann, which is used to create a pipe from a read-only user on the target machine to alter the contents of the cached /etc/passwd file to grant root access. The demo exploit code and instructions on how to successfully perform the exploit are located at the Team B github repository for the CVE20220847 exploit. Additionally, to further support the process of simulating the exploit, the nmap command was used to map out the network from the attacking machine.

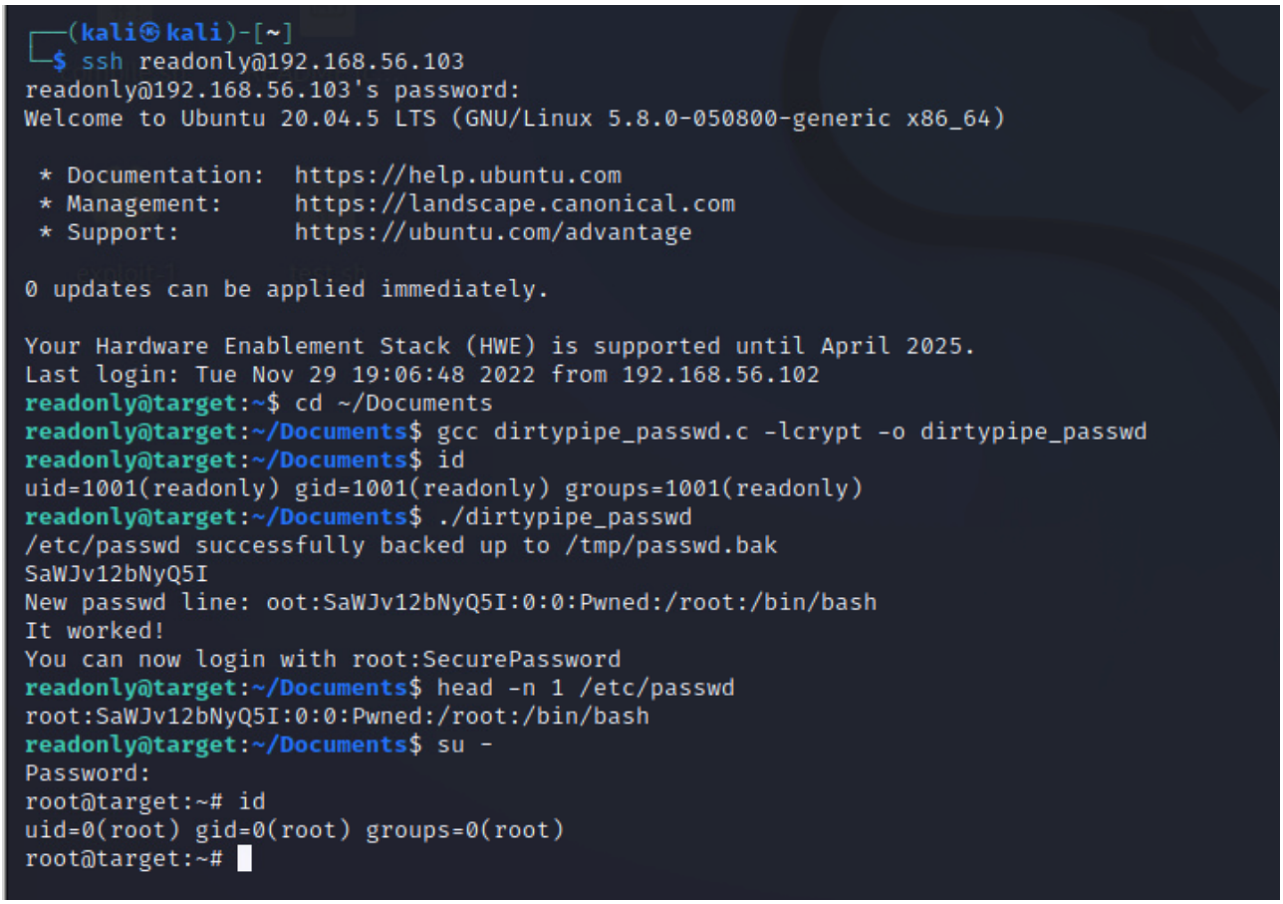

Figure 3: Screen capture from the attacking Kali machine during the execution of the exploit.

Live Detection

Once the exploit was sufficiently demonstrated, Team B moved on to exploring options for live detection. First, from the detection machine, wireshark was used to observe all network traffic across the networked devices. On a larger scale, data collected in this manner could be used for frequent pattern mining to identify abnormal events as it identified bidirectional packets from each machine on the network. However, in this particular instance, the limited data captured did not yield any insight.

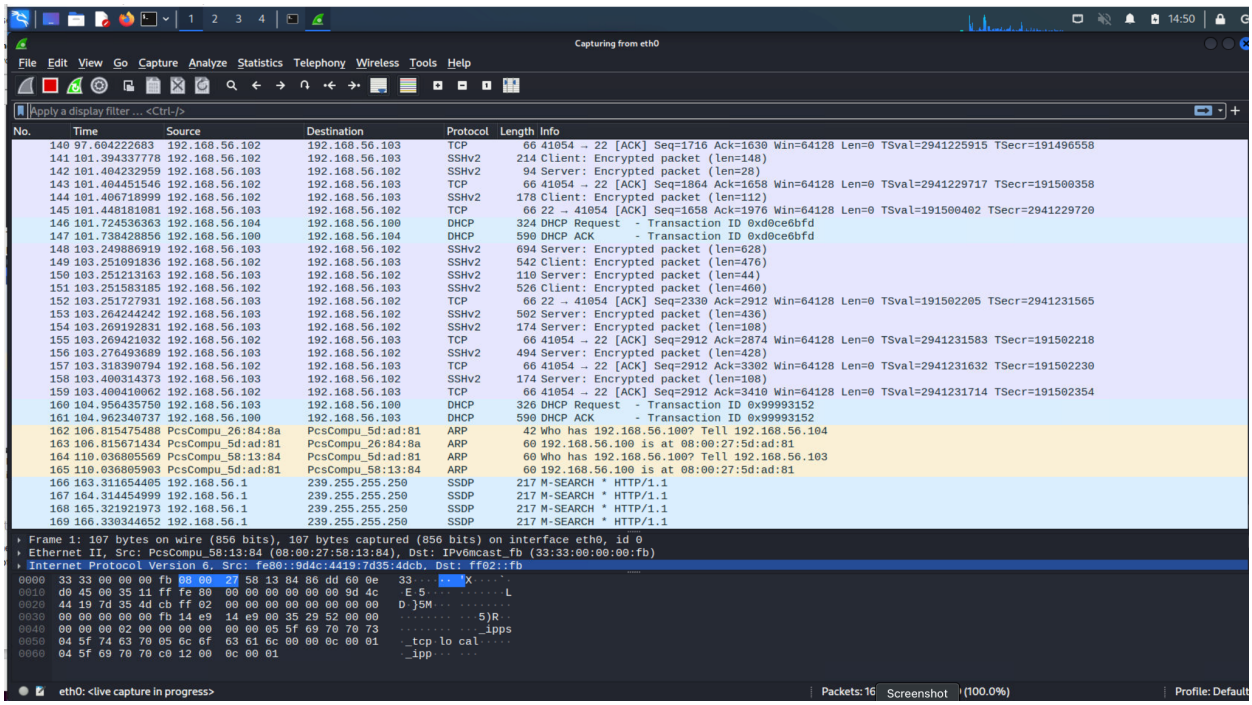

Figure 4: Screen capture from the Wireshark network scanning

To gather more information that can be more readily attached to a particular user's activity on a particular device, live monitoring of commands and files was performed. From the admin account on the target computer, a demonstration C program was written to achieve two objectives:

1. To monitor the /etc/passwd file for any changes between the cache and the /etc/shadow backup file using the `pwck` command.
2. To do a live monitor of all system calls generated by all other users on the target device using the `sysdig` to detect creations of a new root shell, indicating an unexpected escalation to root.

Using these methods, Team B was able to successfully demonstrate the first documented live detection of the CVE20220847 dirtypipe at the time of cache corruption.

A video showing the demonstration of the exploit and live detection, all code, and instructions for compilation can be found on the Team B GitHub repository for the CVE20220847 dirty pipe exploit.

SUMMARY OF LIVE DETECTION

Analysis

We used sysdig to monitor activity for all users on the system for live detection. After downloading the program via `sudo apt-get install sysdig` on the target ubuntu machine, running `w` command all on its own will show all users which may be more informative in other use cases but considering we only have the two users it won't show much more. We can gather more information using the `sudo sysdig -c spy_users` command. We repeated the exploit with sysdig running which allowed us to observe all the commands entered during the exploit.

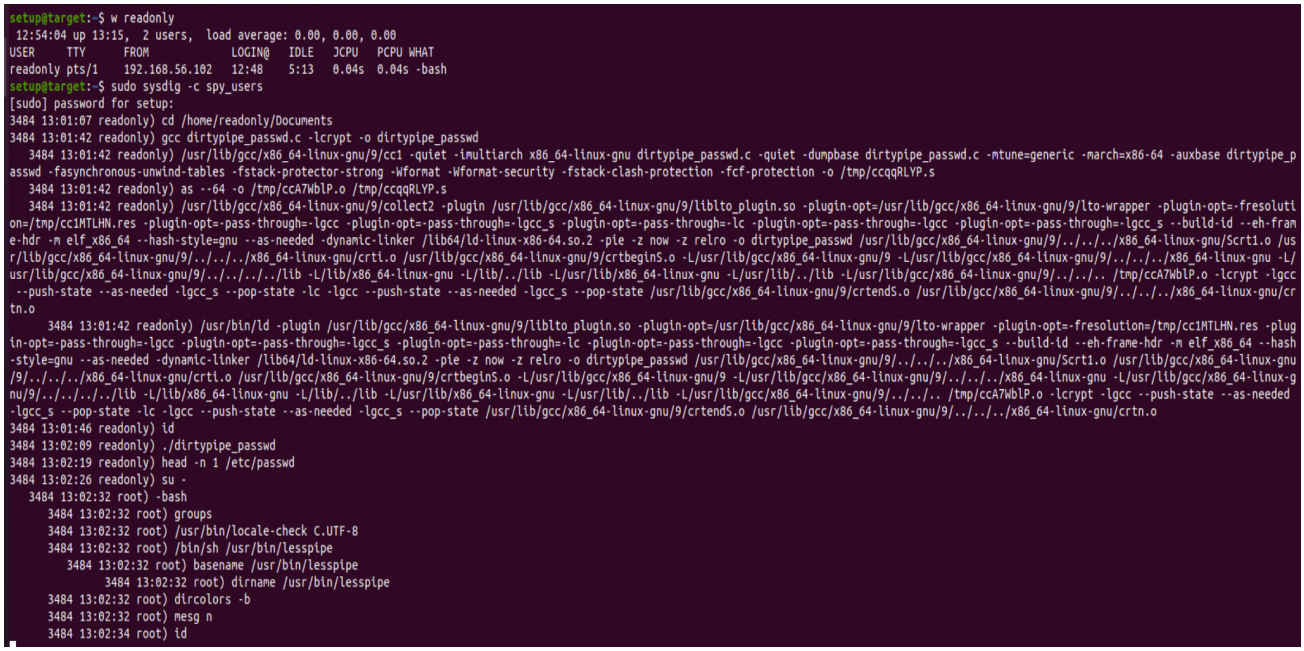

Figure 5: Live Detection Using Sysdig

While the `sysdig` program is running, all commands the user enters while the exploit runs are mapped. First, it shows that the attacker machine which had already established a connection into the non root user via ssh on the Ubuntu machine changed the directory to Documents which is where the dirty pipe exploit's c file is located. The scan then demonstrates how that executable created from the c file is able to change the root password to SecurePassword in the `/etc/passwd` cache, accomplishing its first goal. The second goal of the scanning process is to detect this unauthorized escalation of the root user and the scanning successfully detects that (see screenshot below). The `whoami` command yields the currently logged in user and the `id` command is providing information about the uid, gid, and groups associated with this user. With the readonly user, it is associated with the readonly group which does not have root access for the system. Once the root password is changed via the exploit, the user successfully logs in as root with the

new password and can now make changes to any file or system processes. The live detection piece scans the `/etc/passwd` file for changes and detects a change in the password file and then subsequently, confirms that an unauthorized root escalation has been detected through the unexpected root escalations scan. The second `whoami` command confirms that the logged in user is currently root and no longer readonly.

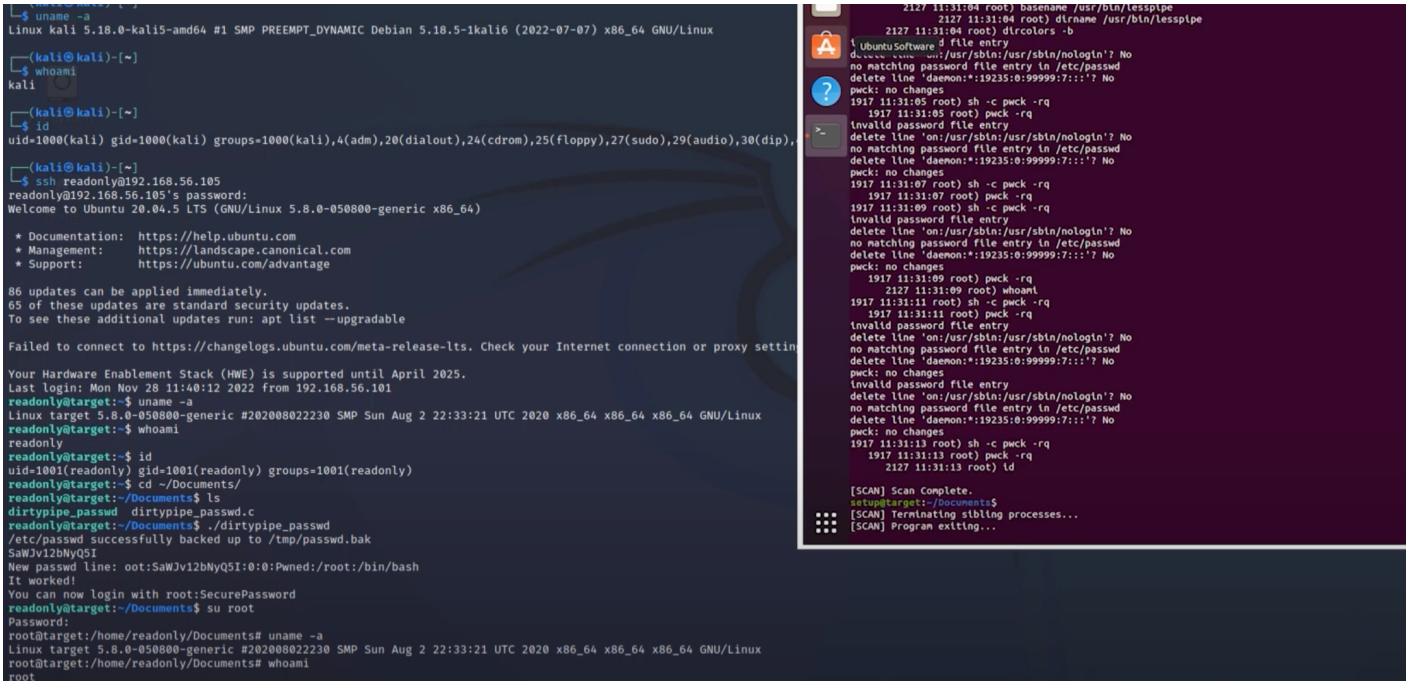

Figure 6: Dirty Pipe Live Detection

Next Steps

The next step in the vulnerability scanning and live detection of the exploit is to evaluate commercial off-the-shelf (COTS) solutions which may also provide similar insights for live detection. One such candidate for evaluation would be Sysdig Secure.

Short Term Recommendations

In the short term, Team B recommends administrators of vulnerable systems scan or use asset management software to create an inventory of all types of Linux kernel running version 5.8 or older within the organization and create a plan to update the kernel to later versions once compatibility with other applications running on the system is verified. If there are compatibility issues, discuss with vendors for possible solutions/workarounds.

Long Term Recommendations

Update your Linux Kernel: Team B recommends that organizations identify all vulnerable systems (Linux versions 5.8 or newer) and update them as soon as possible. The vulnerability has been patched in Linux kernel versions 5.16.11, 5.15.25 and 5.10.102. Updating the kernel will fix this vulnerability.

REFERENCES

DirtyPipe

- https://dirtypipe.cm4all.com/
- https://raxis.com/blog/exploiting-dirty-pipe-cve-2022-0847
- https://github.com/Al1ex/CVE-2022-0847
- https://attack.mitre.org/techniques/T1003/008/
- https://redhuntlabs.com/blog/the-dirty-pipe-vulnerability.html
- https://tryhackme.com/room/dirtypipe
- https://www.youtube.com/watch?v=af0PGYaqIWA

Linux

- https://releases.ubuntu.com/20.04/?_ga=2.158895674.1041833567.1668977229-1417524537.1668977229
- https://www.addictivetips.com/ubuntu-linux-tips/downgrade-ubuntu-kernel/
- https://unix.stackexchange.com/questions/198003/set-default-kernel-in-grub#:~:text=As%20mentioned%20in%20the%20comments,and%20then%20running%20update%2Dgrub%20

VirtualBox

- https://fabian-voith.de/2020/04/21/understanding-virtualbox-networking-schemes-to-set-up-a-good-and-safe-lab/

Root Escalations

- https://biriukov.dev/docs/page-cache/2-essential-page-cache-theory/
- https://manpages.ubuntu.com/manpages/bionic/en/man5/passwd.5.html
- https://manpages.ubuntu.com/manpages/bionic/en/man3/crypt.3.html
- https://www.hackingarticles.in/editing-etc-passwd-file-for-privilege-escalation/
- https://www.networkstraining.com/nmap-scan-ip-range/#:~:text=Simple%20NMAP%20scan%20of%20IP,if%20a%20host%20is%20up

C Programming

- https://faq.cprogramming.com/cgi-bin/smartfaq.cgi?answer=1044654269&id=1043284392
- https://stackoverflow.com/questions/259355/how-can-you-flush-a-write-using-a-file-descriptor
- http://www.crasseux.com/books/ctutorial/Writing-files-at-a-low-level.html

APPENDIX A - TOOLS USED

TOOL	DESCRIPTION
VirtualBox	Version 6.1.36 (macOS), Version 7.0 (Windows), Version 6.1.38 r153438 (Qt5.6.2) (Windows); used as a Virtual machine software to build and deploy virtual machines.
Ubuntu Linux, version 5.8.0-050800-generic	Used as the target machine for the exploit
Kali Linux, version 5.18.0-kali5-amd64	Used as the attacker Virtual Machine and cloned this to use as a detection machine during scanning.
Wireshark	Used for exploitation of vulnerable services and vulnerability scanning.
Nmap, version 7.9.2	Used for scanning ports on hosts.
gcc, version 9.4.0	Used to scan the networks for vulnerabilities.
Sysdig, version 0.26.4	Used to monitor users for Live Detection
pwck	Used to detect changes in the /etc/passwd shadow file, part of shadow-utils 4.8.1

Table A.1: *Tools used during assessment*