Ex070 Submission

Gary Jones

Contents

Executive Summary

Project Overview

A buffer vulnerability was exploited in order to gain access with bash shell. Utilizing nmap a scan discovered Brian's services and by using fuzz testing access was granted through a buffer overflow exploit.

Goals

Risk Ranking/Profile

Critical

Summary of Findings

Through this exercise a critical vulnerability was found which enabled access into the host website enabling the user to rummage throughout the various files in the host network. Through this endeavour a key was found: KEY009-

```
KEY009-=l5\x02&5#\x12s7\x04*3\x08~thh\x0633 zz
```

Risk Rating

The risk of this vulnerability is Critical due to the access granted through the access granted by allowing users to insert any command they desire into the interface and having it run. The only offset to this risk is that sudo is unable to be run thereby granting Administrative access to the system. This is a critical finding that severely compromises the integrity of the system.

Attack Narrative

When analyzing www.artstailor.com with nmap for all open TCP ports port 1337 was found to be open (see figure 1). Utilizing netscape with the command nc the user was able to access and fuzz test the administrator account and thereby taking advantage of a buffer overflow vulnerability (see figures 2 and 3). As seen in figure 3 the user was able to initiate a bash session and gain live access to the file system.

After navigating through the file system the source code was identified (see figures 4, 5, and 6). After reviewing the source code it was found that the vulnerability stems from the interaction of several factors. Within the command block, identified by 'get admin user credential' the command fgets(admin, BUFLEN, stdin); takes the user input and reads up to 1024 characters, as defined by BUFLEN, and passes that into the admin character array which has been defined as size 16. This causes the overflow issue that is exploited by further code explained here. Because this step does not pass the while condition ensuring that the administrator name is brian the admin buffer is flushed with the

command fflush(stdout); this is important because only admin array is reset and all other information is kept.

Based on the above information we transition to the effect with the command blocks identified with 'list available command and Check command against list'. The first block of code is the first area of vulnerability. Due to the buffer overflow causing only the first 16 characters to be erased this block of code is only able to print out a fraction of the available commands available to show. In place of the text it would normally show to the user the program instead shows the text that follows the 16th character the user inserts.

In the last block of text 'Check command against list.' the code only checks that the command the user input matches what is in the list before executing it using the system((commands+CMDLEN*i)); command. Because of the combination of these factors the user is able to type any random series of characters until the 16th spot and then replace the listed commands with one of their choice. In the case of this attack the user utilized the code !/bin/bash to allow direct and live access to the filesystem.

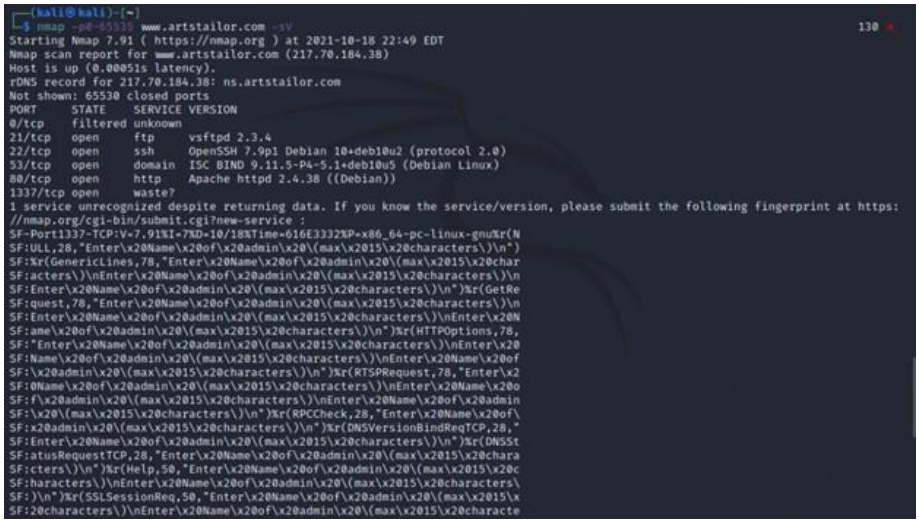

Figure 1:

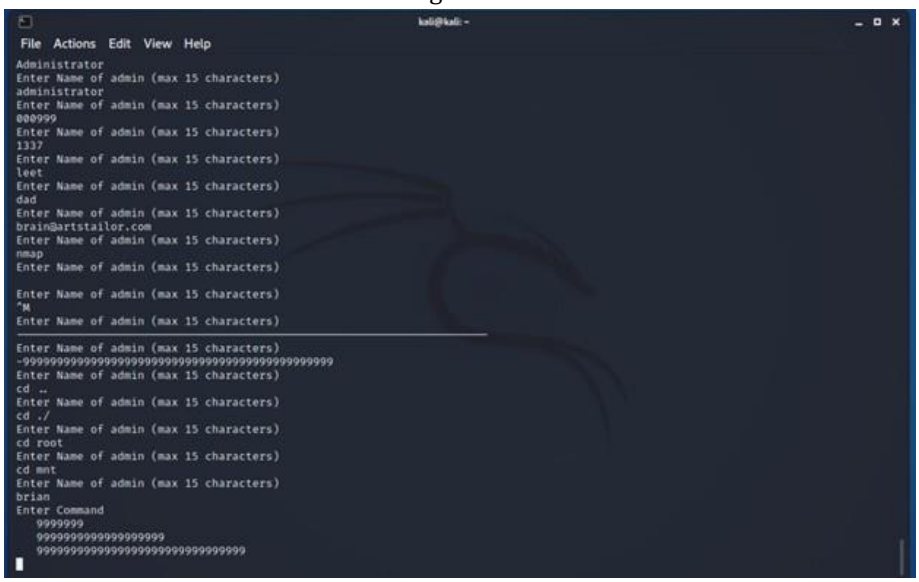

Figure 2:

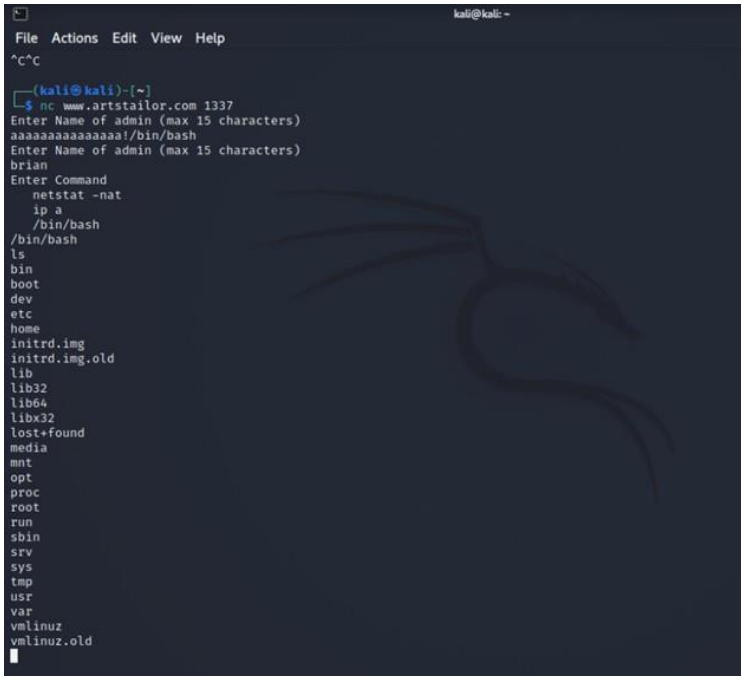
kali@kali: ~
File Actions Edit View Help
^C^C
(kali㉿kali)-[~]
$ nc www.artstailor.com 1337
Enter Name of admin (max 15 characters)
aaaaaaaaaaaaaa!/bin/bash
Enter Name of admin (max 15 characters)
brian
Enter Command
netstat -nat
ip a
/bin/bash
/bin/bash
ls
bin
boot
dev
etc
home
initrd.img
initrd.img.old
lib
lib32
lib64
libx32
lost+found
media
mnt
opt
proc
root
run
sbin
srv
sys
tmp
usr
var
vmlinuz
vmlinuz.old

Figure 3:

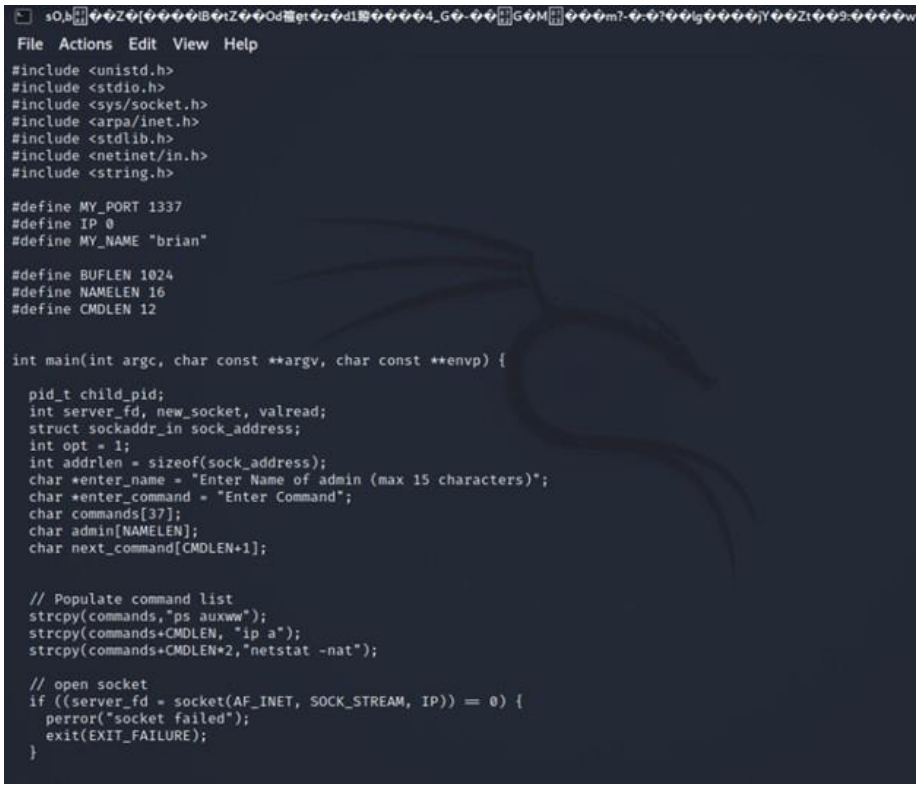

Figure 4:

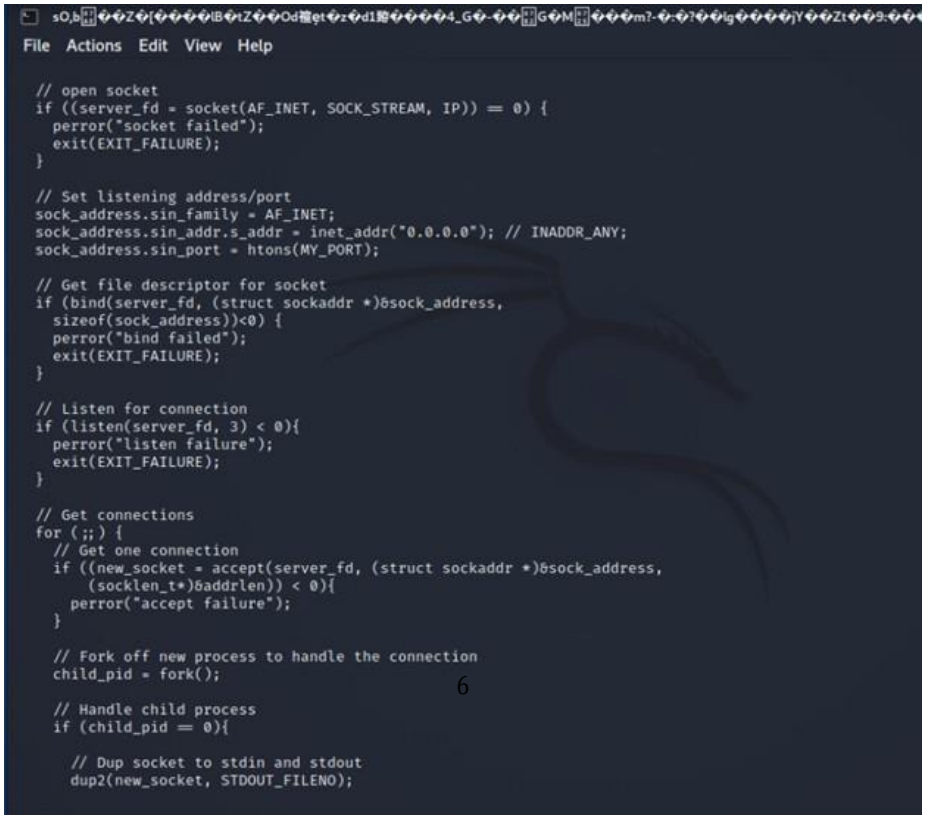

Figure 5:


sO,b
File Actions Edit View Help
dup2(new_socket, STDOUT_FILENO);
dup2(new_socket, STDIN_FILENO);
close(new_socket);
// get admin user credential
while (strcmp(admin,MY_NAME) ≠ 0) {
printf("%s\n",enter_name);
fflush(stdout); // Required for user interaction
fgets(admin, BUFLEN, stdin);
admin[strlen(admin)-1] = '\0';
}
// Process commands
while(1) {
// list available commands
printf("%s\n",enter_command);
for(int i=2; i ≥ 0; i--){
printf(" %s\n", (commands + CMDLEN*i));
}
fflush(stdout);
// read user command, terminate on EOF
if (fgets(next_command, BUFLEN, stdin) == NULL) {
exit(EXIT_SUCCESS);
}
next_command[strlen(next_command)-1] = '\0';
// Check command against list.
// This limits user to our specified command set!
for (int i=2; i ≥0; i--){
if (strcmp((commands+CMDLEN*i), next_command) == 0){
system((commands+CMDLEN*i));
}
}
}
} else {
close(new_socket);
}
}
}

Figure 6: