SIO Project 3 - We were hacked (?)

Licenciatura em Engenharia Informática
Academic Year: 2021/2022
31 January 2022

Element	NMEC
Lucius Vinicius Rocha Machado Filho	96123
Tomé Lopes Carvalho	97939
Dinis dos Santos Lei	98452
Afonso Ribeiro Campos	100055

Executive Summary

Our investigation concluded that we were indeed hacked.
Three attacks were performed. The first one wasn't successful, but, unfortunately, the other two were.
The first was a dictionary attack, which means the attacker attempted to log in as the administrator of our web app using a collection of the most common passwords. It failed because our administrator did not use such a vulnerable password.
The second one was a length extension attack. Through it, the attacker was able to forge an authentication cookie with the administrator's username considered valid by the app.
The last one consisted of code injection. That is, they were able to force our machine to execute their code, because our Flask web application does not implement any mechanism to prevent this. Through this exploit, they were able to:

- Obtain the administrator's credentials (they were hardcoded in the application program and thus the attacker was able to read them once they obtained access to the program's contents)
- Obtain sensitive files such as **/etc/passwd**, which reveals the password for the users of the system.
- Create a backdoor to our system.

Indicators of Compromise

- Log-in Red Flags, for several login attempts with the “admin” username.
- Unusual Domain name requests, with the IP address 192.168.1.122, that corresponds to login on the router as admin on a local area network and are not visible on the internet
- Web Traffic with Unhuman Behavior, used on the Dictionary Attack
- **/etc/crontab** file has been modified.
- **http packages** file, showing the attempts of URL manipulation made

Security Oriented Part

Analysis of modified Data Objects

- **/etc/crontab** file, that he used to create a persistent object.

Analysis of Exfiltrated Data

- /etc/passwd
- /etc/shadow
- app.py
- auth.py
- Environment Variables
- /etc/crontab
- /root/.bash_history
- /root/.ssh/id_rsa.pub
- /home/dev/.ssh/id_rsa
- /home/dev/.ssh/id_rsa.pub
- All files inside /etc/ssl/private/
- All files inside /var/log/
- All logs from the running container

Analysis of Potential Suspect IP Addresses

IP Address	Entity	Motive
192.168.1.122	Attacker (Communication with Server).	The following come from this IP: - Dictionary attack POST requests - Injection GET requests
96.127.23.115	Attacker (IP identified on persistent object). However this address corresponds to Amazon.	This IP is the one that is linked to a persistent object inserted on /etc/crontab, that means, the IP in which it is connected.

Other IP Addresses

IP Address	Entity	Motive
192.168.1.251	Attacked Machine	It's the one replying with HTTP response status codes
199.232.182.132	Debian Repository	Replies to the requests made by the machine due to the "apt update" command executed by the attacker

Indicators of Compromise

MITRE Attack Matrix Mapping

This is important for us to correlate the modus operandi with other attacks. I know we were using some confinement in those VMs, so I'm also curious what could have happened.

Initial Access - Valid Accounts
The attacker obtained the admin credentials to upload a threatening image.

Execution - Command and Scripting Interpreter: Python
The attacker was able to inject python code to interact with the underlying system (Jinja Injection).

Persistence - Scheduled Task/Job - Cron
The attacker appended a line that periodically establishes a reverse bash shell from the victim computer to theirs to the */etc/crontab* file.

Credential Access - Unsecured Credentials: Credentials In Files
The attacker was able to access admin credentials that were hardcoded in app.py.

Credential Access - Brute Force: Password Guessing
The attacker first attempted to log in as the admin using a simple dictionary attack.

Discovery - Permission Groups Discovery: Local Groups
The attacker checks the system's users' permissions.

Exfiltration - Exfiltration Over C2 Channel
The attacker set up a C2 Beacon so he could in the future exfiltrate more data.

Impact - Data Manipulation: Stored Data Manipulation
The attacker uploaded a .png file to the app in order to intimidate and extort the victims.

Initial Access - Exploit Public-Facing Application
Jinja template injection is used by the attacker in order to run commands through the **popen** method from Python's **os** library.

Execution - Deploy Container
The attacker runs busybox containers to run commands.

Reconnaissance - Active Scanning
The attacker got the structure through http requests.

Command and Control - Web service
The attacker tries to cover up its beacon by using the amazon ip 96.127.23.115 to relay data from the system

Privilege Escalation - Escape to Host
The attacker mounted the host's file system on a container using the bind parameter, allowing them to drop payloads and execute control utilities such as cron on the host.

Discovery - File and Directory Discovery
The attacker used the **ls** and **find** commands to enumerate files and directories.

Collection - Data from Local System
The attacker searched local system sources to find files of interest and sensitive data prior to Exfiltration.

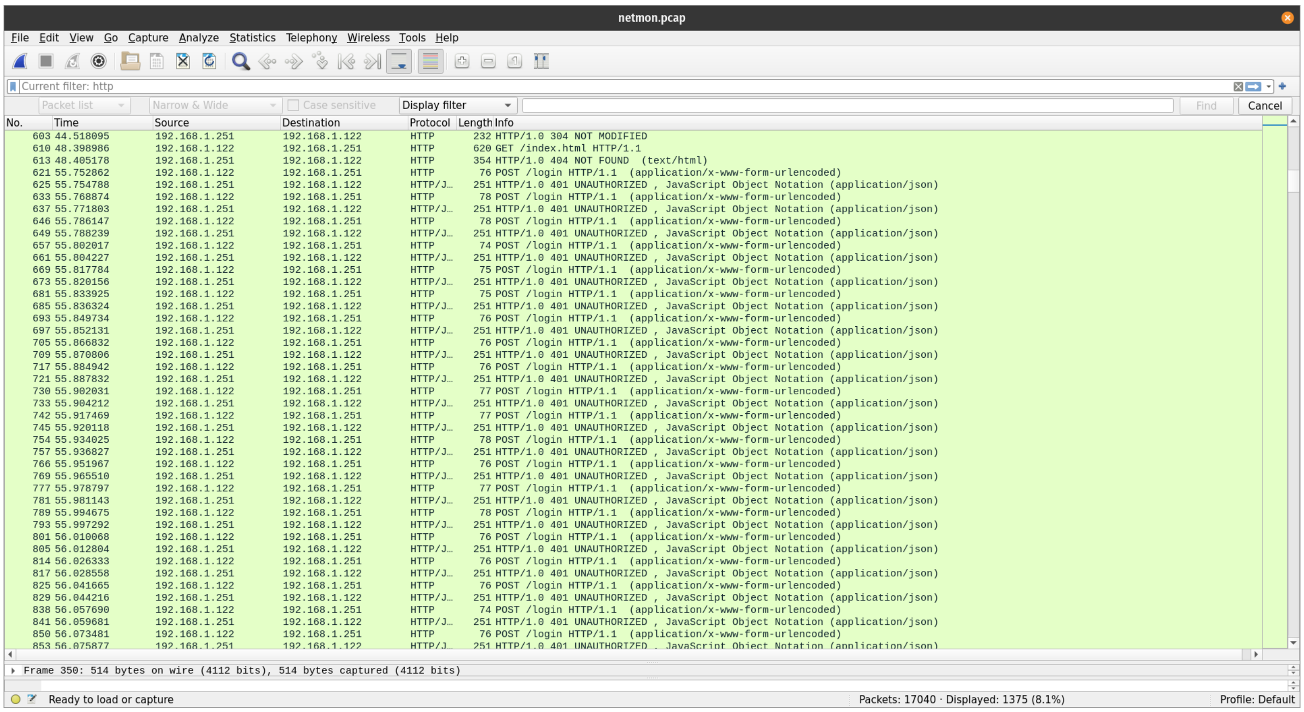
Impact - Defacement
The attacker left a ransom threat in the form of an image on the website.

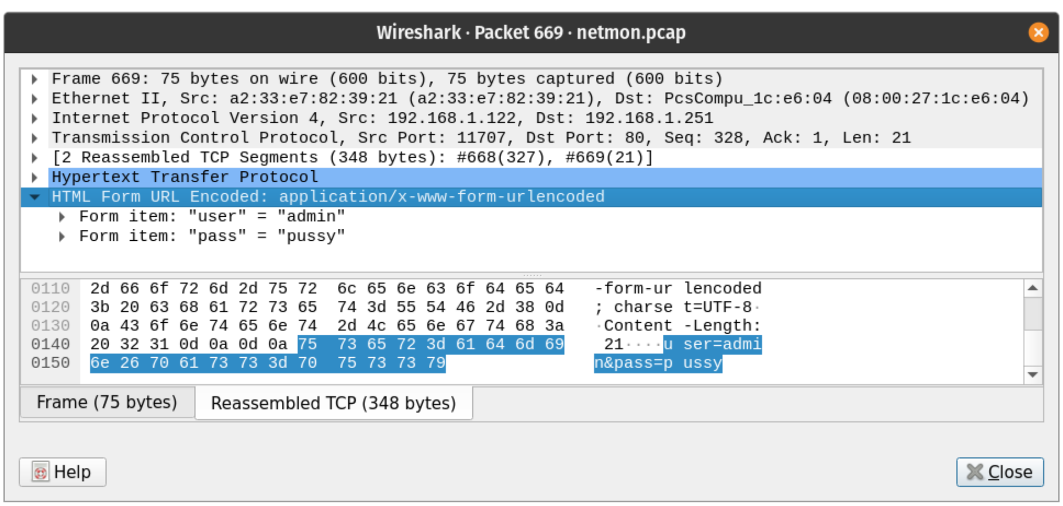
Lateral Movement - Use Alternate Authentication Material
The attacker forged an administrator authentication cookie in order to obtain administrator privileges.

Analysis of the Actions performed by the Attacker

Dictionary attack and start of template injection

The attacker started by attempting a dictionary attack for the username "admin" ([CWE-307: Improper Restriction of Excessive Authentication Attempts]). In layman's terms, they tried to log in using a collection of the most common passwords. No attempt was successful.



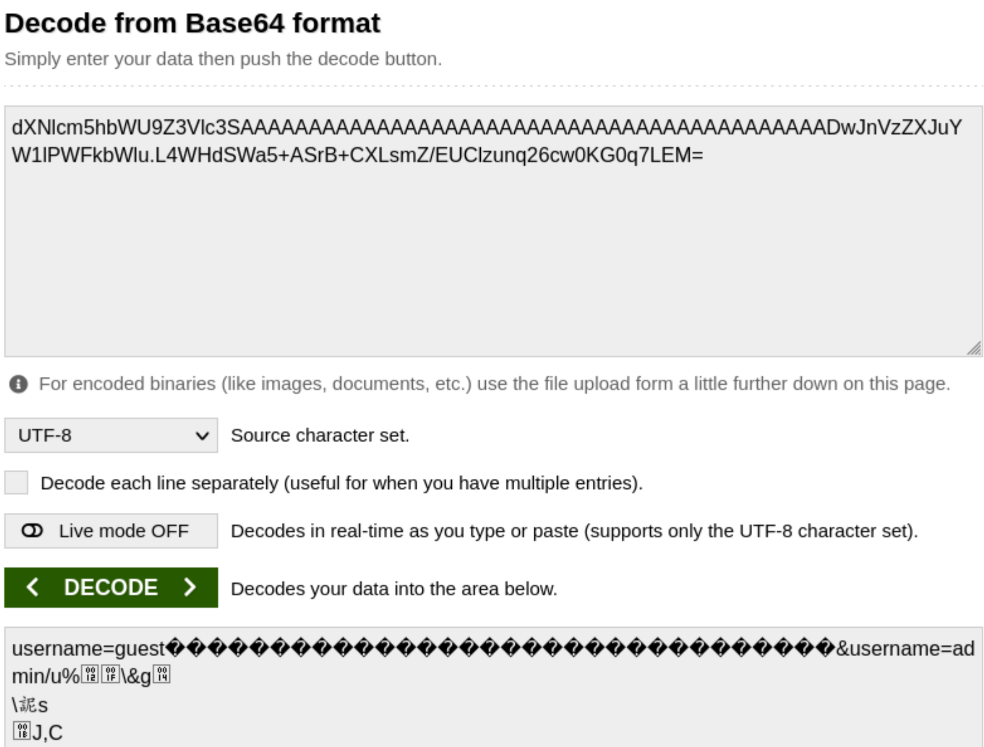


Length Extension Attack

The attacker took advantage of the use of a weak hash (**[CWE-328: Use of Weak Hash]**) and the flexibility in the message format: if duplicate content is in the query string, preference is given to the latter value.

```python
for v in data.split(b'&'):
    kv = v.split(b'=')
    if len(kv) == 2:
        try:
            values[kv[0].decode()] = kv[1].decode()
        except:
            pass
```

In this code block from the auth.py file, **values** is a dictionary.
The code does not check if, in each iteration, the key already exists in the dictionary.



This way, the attacker was able to make **values**[‘username’] equal to **admin**.
They performed a [Length Extension Attack], using the signature of a valid guest cookie returned by the server as an original cookie. They could then initialize a hashing algorithm, input the last few characters, and generate a new digest which can sign their new message without the original key (which is, in the case of our app, randomly generated each time it’s run).

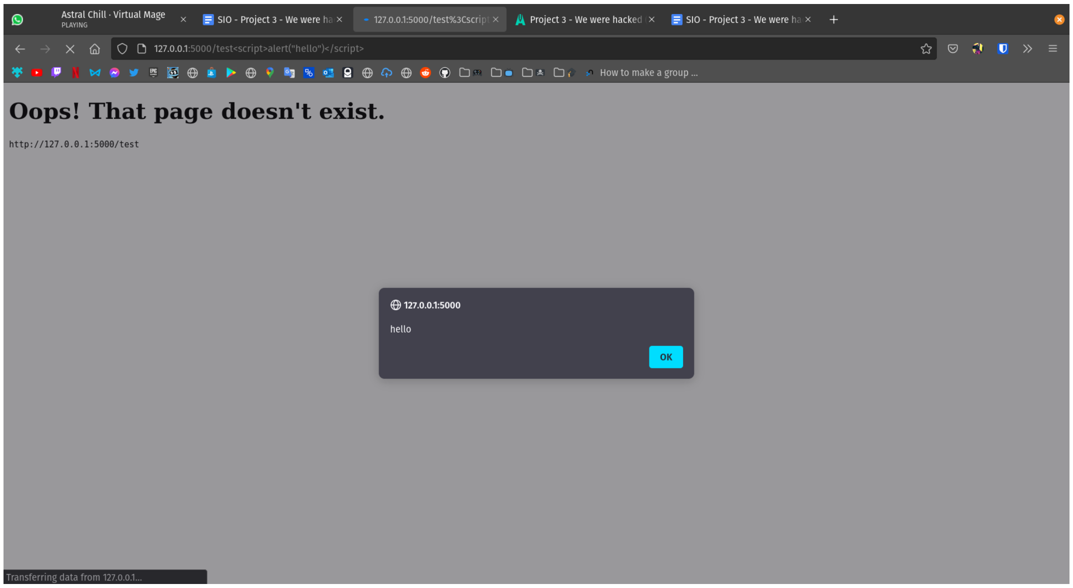
```
HTTP/1.0 200 OK
Content-Length: 6195
Set-Cookie: auth=dXNlcm5hbWU9Z3Vlc3Q=.IaRReH75V/N0jyWcxFdIo0qIeNhhC51JqV3SHTH0nJo=; Path=/
Access-Control-Allow-Origin: *
Server: Werkzeug/2.0.2 Python/3.9.5
Date: Thu, 06 Jan 2022 19:16:10 GMT
E.....@...c....z..../..PN...v.7UP. ...

GET

HTTP/1.1
Host: 192.168.1.251
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/96.0.4664.110 Safari/537.36
Accept-Encoding: gzip, deflate
Accept: */*
Connection: keep-alive
Cookie: auth=dXNlcm5hbWU9Z3Vlc3SAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAADwJnVzZXJuYW1lPWFkbWlu.L4WHdSWa5+ASrB+CXLsmZ/EUClzunq26cw0KG0q7LEM=
E..9h.@.?.N1.......z.P/.v.7UN...P.....

HTTP/1.0 200 OK
Content-Length: 6053
Access-Control-Allow-Origin: *
Server: Werkzeug/2.0.2 Python/3.9.5
Date: Thu, 06 Jan 2022 19:16:10 GMT
E.....@...c....z..../..P...KL.BcP. .(.
```

The first block in this image represents an HTTP response with Set-Cookie. This happens when the server concludes the user doesn't have a valid authentication cookie. As we can see, the second response (to the GET request in the middle) does not have Set-Cookie. We can thus infer that the forged cookie was accepted by the server, allowing the attacker to obtain administrator privileges.

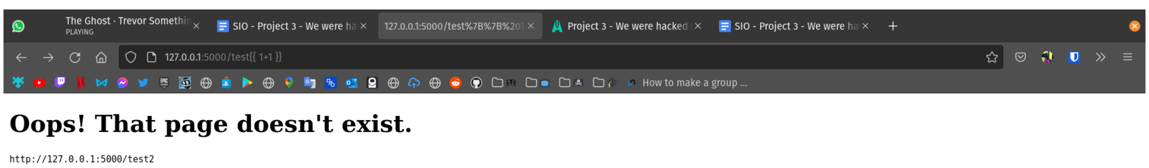
Following this event, the attacker now possessing administrator privileges, tested some URLs, such as "/private", but they didn't exist.

Jinja Template Injection Attack

They experimented with injection through GET requests, showing the results in the page users are redirected to when the URL they request is not found. The first XSS injection (**CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')**) tested used JavaScript code: a simple *alert* function call.
The test demonstrated that the application is, in fact, vulnerable to such attacks.



The attacker then discovers the application is vulnerable to ***Jinja Template Injection*** ([CWE-1336: Improper Neutralization of Special Elements Used in a Template Engine]) by injecting a trivial sum operation. The application proved to be vulnerable once again.



Notice the 2 shown after the URL. It's the result of the 1 + 1 operation injected.

With code injection being a valid option, the attacker then inserts Python code that allows them to access the os library: **request.application.__globals__.__builtins__.__import__('os')**. This makes it possible for the attacker to exploit its [popen] method, which opens a pipe to or from the command *cmd*, allowing them to execute shell commands.

The attacker continued to experiment with the commands **id** (shows usernames, IDs, groups) and **ls**, which allowed them to discover the application root files. Next, using the **cat** command, they print the content of the files **app.py** and **auth.py**, discovering the **hardcoded admin credentials ([CWE-798: Use of Hard-coded Credentials]).** Afterwards, he reads the content from /etc/passwd and /proc/mount, but not /etc/shadow file, due to lack of permission.

Later, they listed the system's files with the command **find /**, created a new file named **.a** on the app directory and executed the **ls -la .a** command to check the new file's permissions (possibly to assess what the default permissions were).

They then attempted to list the files (and their permissions) in the **/root/** folder, but were not able to because accessing this folder requires superuser privileges. Next, they checked the files residing in **/home/**.

After that, they tried to show the files that had setuid permissions with the command **find / -perm -4000**, but again were denied permission for plenty of the results. Lastly, they used the **env** command to list all the environment variables.

Docker Containers

The attacker listed the docker containers using **docker ps**. Then, they proceeded to install an **earlier version of Docker, with the package name docker.io** through the **apt update** and **apt install -y docker.io** commands.

From this moment, almost all their docker actions involve running a **busybox** image and mounting the root files inside of its volume for later access or executing a command using the **docker run --rm -t -v /:/mnt busybox** suffix ([CWE-288: Authentication Bypass Using an Alternate Path or Channel]). The first of these attempted actions was mounting the root to the /mnt folder and doing the **find /mnt/** command to see the files of the mounted root inside the docker.

Next, they executed Python code which appends a line to the mounted **/etc/crontab** file, adding a **persistent C2 Beacon object**, creating a **reverse shell for the IP 96.127.23.115** (IP Address corresponding to Amazon) every 10 minutes (meaning that even the reverse shell file is deleted for some reason, it'll be created again).

They then proceed to read the **bash_history**, the **ssh public key**, **/etc/passwd**, **/etc/shadow**, **ssl** (certificates), all logs inside **/var/log** and the container json log files, as well as the files in **/home**. He also tried to get config files inside **/etc/mysql**, but those don't exist.

Ending the attack

After successfully stealing the information, the attacker uploaded the image containing the ransom threat using the admin account and restarted the app Docker container, with the objective of clearing their traces.

Conclusion - Potential Intentions and Impact Mitigation

The objective of this attack was to steal important user data (passwords) and use it as ransom, as well as set up a backdoor for future attacks.
To mitigate the impact of the attack, an email should be sent to all users warning that their passwords may be compromised and should be changed immediately.
A full OS wipe and reinstall could be done to safely guarantee that no malware remains on the infected machine. It'd be wise to make sure no credentials from the previous install are reused.

The following measures should be taken in order to eliminate vulnerabilities:

- Change compromised admin account password
- Save passwords in a secure way instead of hard-coding them
- Perform input sanitization to prevent code injection attacks
- Use the HMAC hash algorithm to create auth cookie signatures instead of sha256

If the machine is not wiped:

- Change the Debian machine's passwords
- Eliminate the line appended to ***crontab*** by the attacker