Project 3

Summary

Following an attack to the services of UA Pintaresto, the team was solicited to preform an analysis in an attempt to bright to light in what way the services were exploited and what data could have been stollen and/or destroyed.

Description

Following detection of an intrusion, by the automatic systems employed by UA Pinteresto, the team responsible for security of this company was not certain of what may, or may not, have happen to they're systems.

What this team offers as information is that the VM in service was stopped and recorded a **PCAP file** with the past **traffic to and from the VM**.

For analysis, the UA Pintersto's team offer the following data:

1. Capture of the traffic by the VM in the form of:

    a. PCAP file - **netmon.pcap**

    b. Text file containing the HTTP packages of said capture - **http.txt**

2. A **virtual disk image**, of the operating VM, containing sensitive data about the web application of UA Pintaresto

A careful analysis was preformed in each single file, as to leave no doubts or any unchecked piece of data.

Exploration and Discovery

Do to the fact that the provided data for analysis is not overwhelming, a file by file discovery was the first step in a multilayered analysis.

This said, the file by file order was the following:

1. netmon.pcap
2. http.txt
3. VDI

This order is not random. Being this a internet service, all the data is transmitted via this mean, and so we found it best to focus our initial efforts in discovering **what was transmitted** between the service and a potential attacker.

This way, we could use de traffic capture file as a sort of a chronological timeline of the entire attack. This proved to be a very fruitful method, and guided our analysis the entire way.

The **http.txt** was mainly used as a mean to further explore the HTTP data exchanged between attacker and server.

Attacker Actions and Behaviour

To describe the action taken by the attacker, we'll resort to the **MITRE ATT&CK** matrix, witch well describes the complete set os procedures that can used by an attacker to execute a successful attack.

This stages are ordered chronologically.

Reconnaissance

Upon analysis of the traffic captured in the **netmon.pcap** file, a suspicious IP (192.168.1.122) was detected.

This IP is recorded to navigate between the main and the upload pages several times.

At one point, indicated by the package number **445** and **455** of the capture, this IP tries to login with the usernames, **admin** and **guest** and what appears to be a random password, witch results in unauthorized messages.

Credential Access

- **Brute Force**

  After the previous unsuccessful attempts, a **brute force** attack, against the **admin** username, is initiated by the same IP address (package number **621**).

  This attack lasted until package number **6613**, accounting for a total o **2996** password attempts.

  Each one of this attempts resulted in **failure**.

  Prevention

  Prevention agains brute force attacks relay heavily on restraining the speed, limit and form of attempts of authentication.

  - **Speed**

    One simple way of limiting the speed of the attempts is to create the illusion of loading time of login, this can severely decrease the number of *attempts per minute*, witch will discourage the attacker.

  - **Limit**

    Perhaps the easiest way to prevent this type of attack is to set a limit of attempts per IP address or username. If the limit is met, no more attempts will be allowed, and the user will be notified with a url to reactivate the account's login or change the password.

  - **Form**

    By using **CAPTCHA** or some sort of **two factor authentication (2FA)**, any login attempt will only be complete with a human interaction (in the case of CAPTCHA) and thus preventing

programmable and automatic attempts, or with a completely separate authentication device of software (2FA), thus preventing the possibility of successful match with the brute force dictionary resulting in full access to a user's account by the attacker.

- **Credential Forgery**

  Since the privious method was proven unsuccessful, the attacker gives un in trying o login into the service, and will try to **hijack the session** of any already logged in user.

  For this, the attacker begins (in package number **6658**) a series of ***client-side cookie poisoning*** attacks, in an attempt of a match with any user's cookies an thus hijacking said user's session.

  This attack lasted until package **7758**, totalling **50** attempts of *cookie poisoning*.

  This technic also resulted in **failure**.

  Prevention

  Although none of the attackers attempts to match any of the existing cookies, this could very well have happened. A major security flaw of this service, is the use of **persistent cookies**. This means, that even if a user leaves the website, the cookie remains *alive*, and if captured, could be used to gain access to that user's account.

  This vulnerability can be further explain in the **CWE-539: Use of Persistent Cookies Containing Sensitive Information**

  The prevention for this type of vulnerability passes by implementing **secure session cookies**, ensuring the cookies are unusable once the session is closed. By having comprehensive session management, per example, the service is lacking a way of logging out, one can increase the protection against various attacks, The emplementation of temporary cookies set to expire in a certain time span would also increase changes to resist attacks like *cookie stealing* and *cookie poisoning*.

Search Open Websites/Domains

Having failled to gain access to the service via exploiting third party user's credentials or tokens, the attacker began a short reconnaissance in search for open subdomains of the service.

The attacker experimented with names like "**/private**", "**/fdssfdf**" and "**/test**".

Initial Access

Exploit Public-Facing Application

Having settle with the **/test** page, the attacker began to attempt to inject code into the service. The attacker experiments with **Reflected Cross-Site Scripting**, witch is defined via the **CWE-79: Improper Neutralization of Input During Web Page Generation** passing scripting code through *GET Request* via the URL input

This attack was **successful** !

The fact that this attack worked, exposed the **biggest flaw** of the service, this flaw is located in the **app.py** file, witch is responsible for serving the webpages and responding to *HTTP Request*.

In this file, the following method is trioggered when a ***404 HTTP Error*** **(Page not found)** occurs

```
@app.errorhandler(404)
def page_not_found(e):
  template = '''
<div class="center-content error">
<h1>Oops! That page doesn't exist.</h1>
<pre>%s</pre>
</div>
  ''' %  (urllib.parse.unquote(request.url)) # <-- !! Unsanatized input !!
  return render_template_string(template, dir=dir, help=help, locals=locals), 404
```

As we can see, and pointed out by the comment, **whatever input** may have been passed as a URL during the *GET Request* that resulted in a *404*, will be **directly** inserted in the HTML that will be returned and displayed to the client.

Additionally, due to the format chosen to insert python variables inside the *string* data, when **double curly braces** are used, the text taken inside the pair, is considered as **pure code** and thus, it **is executed as part of the script**.

Prevention

The best way to prevent attacks by cross-site scripting is to sanitize any data that is inputed by the user, but no only that, if you need to combine it with other commands, or it will be placed in sensitive location, the form in witch you combine the hardcoded data with the data provided by the user must be secure in a way that converts this dynamic data in pure text, and cannot be interpreted as code to be executed. In few words, all the data provided via text input by the user, must be recognized as a string.

In this particular case, since the service uses Flask, one can utilize the **flask.escape()** to sanitize the input against HTML/Javascript code insertion.

For sanitization, since there are no pages thar require especial characters, this characters can be filtered out as to not cause any disturbance in the code. This is what's demonstrated in the adapted code bellow.

```
app.errorhandler(404)
def page_not_found(e):
    template = '''
 <div class="center-content error">
 <h1>Oops! That page doesn't exist.</h1>
 <pre>%s</pre>
 </div>
 '''% urllib.parse.unquote(request.url).translate ({ord(c): " " for c in "!@#$%^&*
()[]{};:,./<>?\|`~-=_+"})

    return render_template_string(template, dir=dir, help=help, locals=locals),
404
```

Discovery

Now that the attacker can control the website, the focus now lays in trying to explore the server system and figure out what data it can be exfiltrated.

In the process of looking into critical data like the contents of **/etc/shadow** and **/etc/passwd**, and by being able to run commands like **"apt update"**, the attacker now also knows it has **super user** privileges, witch must have been inherited by the **app.py** being put into production under super user access. If this hadn't been done, all of this capabilities would have been negated to the attacker, unfortunately this was not the case.

Deploy Container

The next step for the attacker, was to install Docker. This would prove to be very usefull since it will allow him to install certain tools to further exploit the server.

The Docker container will allow the user to install and use BusyBox in the next steps.

Software Deployment Tools

As said before, the attacker will use BusyBox for the next steps, but what is BusyBox ?

```
BusyBox combines tiny versions of many common UNIX utilities into a single small
executable. It provides replacements for most of the utilities you usually find in
GNU fileutils, shellutils, etc. The utilities in BusyBox generally have fewer
options than their full-featured GNU cousins; however, the options that are
included provide the expected functionality and behave very much like their GNU
counterparts. BusyBox provides a fairly complete environment for any small or
embedded system.

BusyBox has been written with size-optimization and limited resources in mind. It
is also extremely modular so you can easily include or exclude commands (or
features) at compile time. This makes it easy to customize your embedded systems

in https://busybox.net/about.html
```

With BusyBox the attacker has access to a full suite of utilities it may need to continue to exploit the system.

Persistence

When it comes to set a way for future access, the attacker will use a *Boot or Logon Autostart Execution* technique, particularly by setting up a **reverse shell** (can be seen in package **24745** of netmon.pcap) to the IP/Port of **96.127.23.115/5556**. This is possible by adding the command that creates this connection to the **crontab** file, that executes a specific command in a specific time windows, in this case, the attacker set it to be executed whenever possible.

Exfiltration

As to the data the attacker may have stollen, the actions taken by it indicate that the **passwd** and **shadow** file were explored. Not only that, more importantly, also the **RSA keys for the SSH** tunnelling and the private **SSL certificates** are also exposed and potentially stollen.

Impact

In the end, by having exposed all of this data, and now knowing the **admin's password**, since it is **hardcoded in app.py** the attacker placed a images demanding a ransom to be paid in the value of **100 bitcoins** with the threat that, know having access to the system permanently via the reverse shell, the attacker can delete whatever data it choses. Finally, the user restart the docker application.

Vulnerabilities

This is just a short list of all the vulnerabilities previously mentioned. See the previous explanations to see where they exist and how they can be mitigated.

CWEs Founded

1. **CWE-539: Use of Persistent Cookies Containing Sensitive Information**
2. **CWE-79: Improper Neutralization of Input During Web Page Generation**

The Team

Nome	Número Mecanográfico
David José Araújo Ferreira	93444
Ana Catarina Correia Filipe	93350