JSON - 7th Dec 20

10.10.10.158

Scanning

We run `masscan_to_nmap.py`, a tool I made that you can find on my Github. It runs a Masscan, identifies open ports, and then takes those open ports over to Nmap, and scans for versions and default scripts against those ports.

```
purp1ew0lf@Kali:~/Downloads/json$ sudo python3 masscan_to_nmap.py -i 10.10.10.158
[sudo] password for purp1ew0lf:

Running Masscan on network tun0 against the IP 10.10.10.158 to quickly identify open ports

Starting masscan 1.0.5 (http://bit.ly/14GZzcT) at 2020-12-07 15:44:29 GMT
 -- forced options: -sS -Pn -n --randomize-hosts -v --send-eth
Initiating SYN Stealth Scan
Scanning 1 hosts [131070 ports/host]

Running Nmap full nmap scan against 10.10.10.158 with the following ports 49158,137,49152,49153,5985,49156,21,445,49154,80,49157,49155,135,47001,139,

Host discovery disabled (-Pn). All addresses will be marked 'up' and scan times will be slower.

Nmap results saved to nmap_10.10.10.158.txt
```

```
PORT      STATE  SERVICE      VERSION
21/tcp    open   ftp          FileZilla ftpd
| ftp-syst:
|_  SYST: UNIX emulated by FileZilla
80/tcp    open   http         Microsoft IIS httpd 8.5
| http-methods:
|_  Potentially risky methods: TRACE
|_http-server-header: Microsoft-IIS/8.5
|_http-title: Json HTB
135/tcp   open   msrpc        Microsoft Windows RPC
137/tcp   closed netbios-ns
139/tcp   open   netbios-ssn  Microsoft Windows netbios-ssn
445/tcp   open   microsoft-ds Microsoft Windows Server 2008 R2 - 2012 microsoft-
5985/tcp  open   http         Microsoft HTTPAPI httpd 2.0 (SSDP/UPnP)
|_http-server-header: Microsoft-HTTPAPI/2.0
|_http-title: Not Found
47001/tcp open   http         Microsoft HTTPAPI httpd 2.0 (SSDP/UPnP)
|_http-server-header: Microsoft-HTTPAPI/2.0
|_http-title: Not Found
49152/tcp open   msrpc        Microsoft Windows RPC
49153/tcp open   msrpc        Microsoft Windows RPC
49154/tcp open   msrpc        Microsoft Windows RPC
49155/tcp open   msrpc        Microsoft Windows RPC
49156/tcp open   msrpc        Microsoft Windows RPC
49157/tcp open   msrpc        Microsoft Windows RPC
49158/tcp open   msrpc        Microsoft Windows RPC
```

Enumeration

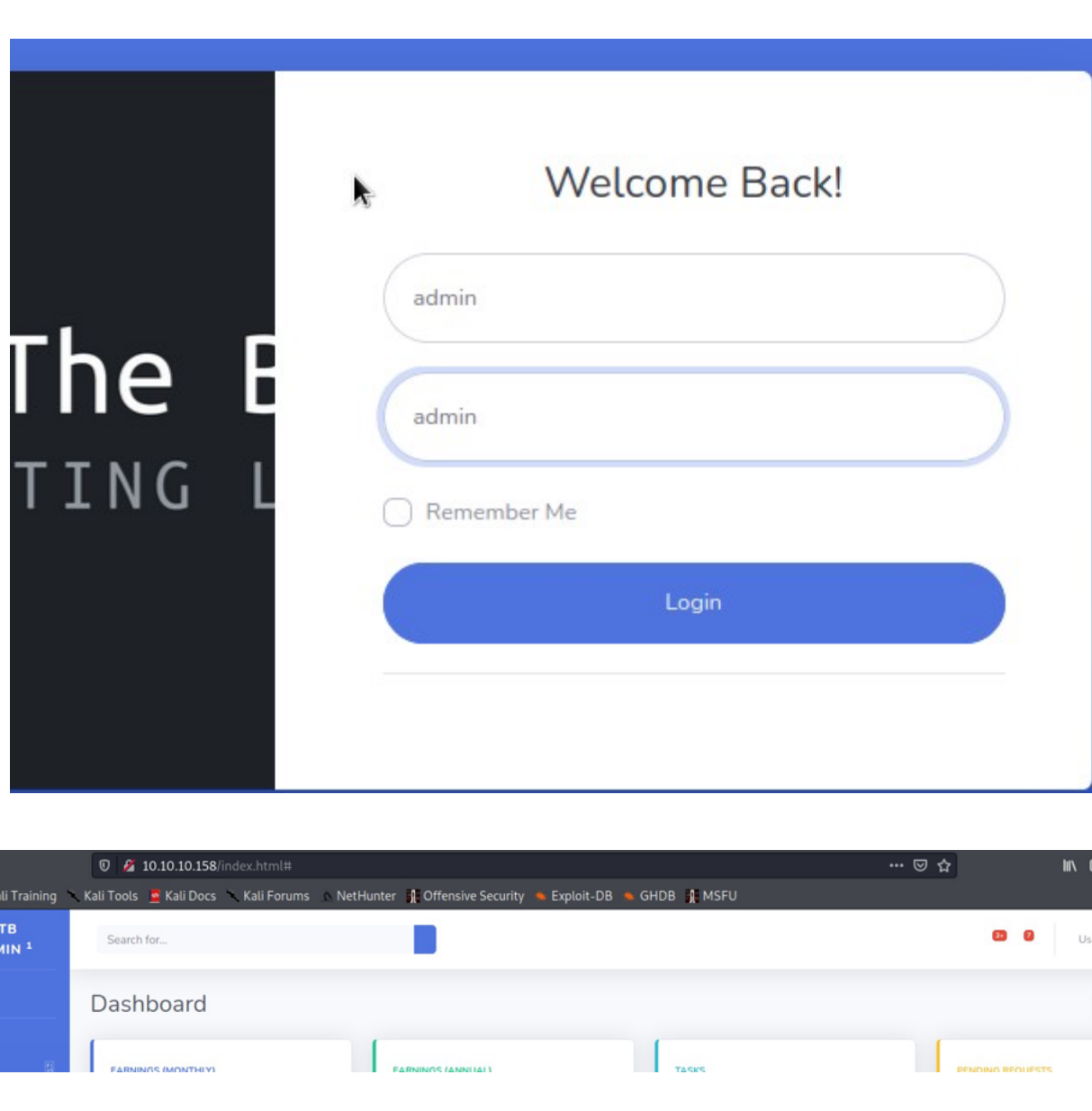
If we checkout the SMB ports with `enum4linux` and `smbmap` we don't get access to anything

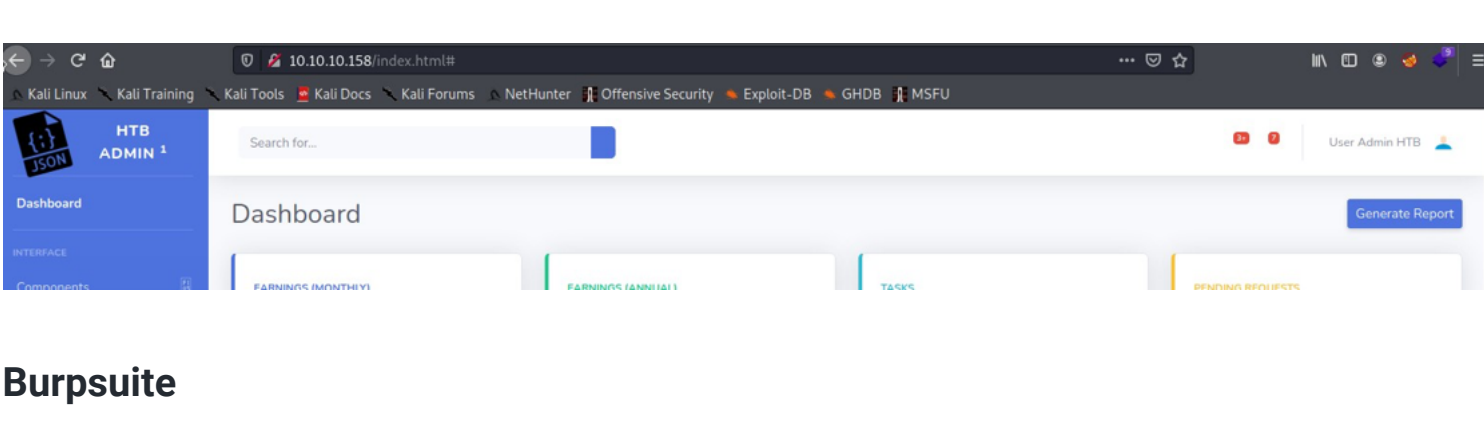
```
purp1ew0lf@Kali:~/Downloads/json/enum/enum4linux-ng$ enum4linux -a 10.10.10.158 -u 'guest' -p 'guest'
Starting enum4linux v0.8.9 ( http://labs.portcullis.co.uk/application/enum4linux/ ) on Mon Dec  7 15:54:27 2020

 ==========================
|    Target Information    |
 ==========================
Target ........... 10.10.10.158
RID Range ........ 500-550,1000-1050
Username ......... ''
Password ......... ''
Known Usernames .. administrator, guest, krbtgt, domain admins, root, bin, none

 ====================================================
|    Enumerating Workgroup/Domain on 10.10.10.158    |
 ====================================================
[+] Got domain/workgroup name: WORKGROUP

 ============================================
|    Nbtstat Information for 10.10.10.158    |
 ============================================
Looking up status of 10.10.10.158
        JSON            <00> -         B <ACTIVE>  Workstation Service
        WORKGROUP       <00> - <GROUP> B <ACTIVE>  Domain/Workgroup Name
        JSON            <20> -         B <ACTIVE>  File Server Service

        MAC Address = 00-50-56-B9-9A-39
```

The FTP port doesn't have anything good for us either. So let's move on to enumerating the website in port 80.

Website

The page starts to load but we get re-directed to a log in. Trying just admin;admin works, even though it asks for an email address but okay?





Burpsuite

If we look at Burpsuite, we find somthing quite interesting. There's an API that calls on the Account and creates a base64 cookie from our input

```
GET /api/Account/ HTTP/1.1
Host: 10.10.10.158
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101 Firefox/78.0
Accept: application/json, text/plain, */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Bearer: eyJJZCI6MSwiVXNlck5hbWUiOiJhZG1pbiIsIlBhc3N3b3JkIjoiMjEyMzJmMjk3YTU3YTVhNzQzODk0YTBlNGE4MDFmYzMiL
Connection: close
Referer: http://10.10.10.158/index.html
Cookie: OAuth2=eyJJZCI6MSwiVXNlck5hbWUiOiJhZG1pbiIsIlBhc3N3b3JkIjoiMjEyMzJmMjk3YTU3YTVhNzQzODk0YTBlNGE4MD
```

If we decode the base64 cookie, it shows us our original admin;admin

```
HJhdG9yIn0=" | base64 -d
{"Id":1,"UserName":"admin","Password":"21232f297a57a5a743894a0e4a801fc3","Name":"User Admin HTB","Rol":"Administrator"}
```

De-Serialisation

Bearer

I don't know what **bearer** is in the intercepeted burpsuite request. Reading around, it seems to be a **java** serialisation thing, which suggests that it may be vulnerable to de-serialisation attacks

- https://swagger.io/docs/specification/authentication/bearer-authentication/
- https://stackoverflow.com/questions/40375508/whats-the-difference-between-jwts-and-bearer-token

Messing around with payloads in the **bearer** section reveals information that can help put our exploit together

```
Server: Microsoft-IIS/8.5
X-AspNet-Version: 4.0.30319
X-Powered-By: ASP.NET
Date: Mon, 07 Dec 2020 20:39:50 GMT
Connection: close
Content-Length: 145

{
  "Message":"An error has occurred.",
  "ExceptionMessage":"Cannot deserialize Json.Net Object",
  "ExceptionType":"System.Exception",
  "StackTrace":null
}
```

De-serialisation attacks