

Mirai

Ursa



Contents

1	Overview	2
2	Walkthrough	3
2.1	Scanning	3
2.2	Gaining Access	4
2.3	Maintaining Access	4
2.4	Escalating Privileges	5
2.5	Covering Tracks	7
3	Conclusion	7
3.1	Final thoughts	7
3.2	Tracking & Reporting	8

1 Overview

Mirai is a challenge machine that emulates the discovery and exploitation of IoT devices on home networks. At the time of this writing, there are more than enough user and root owns, so this writeup will serve as an illustrative guide for the beginner to improve their workflow and I'll include a minimal "project document" at the end that shows the process of documenting the engagement.



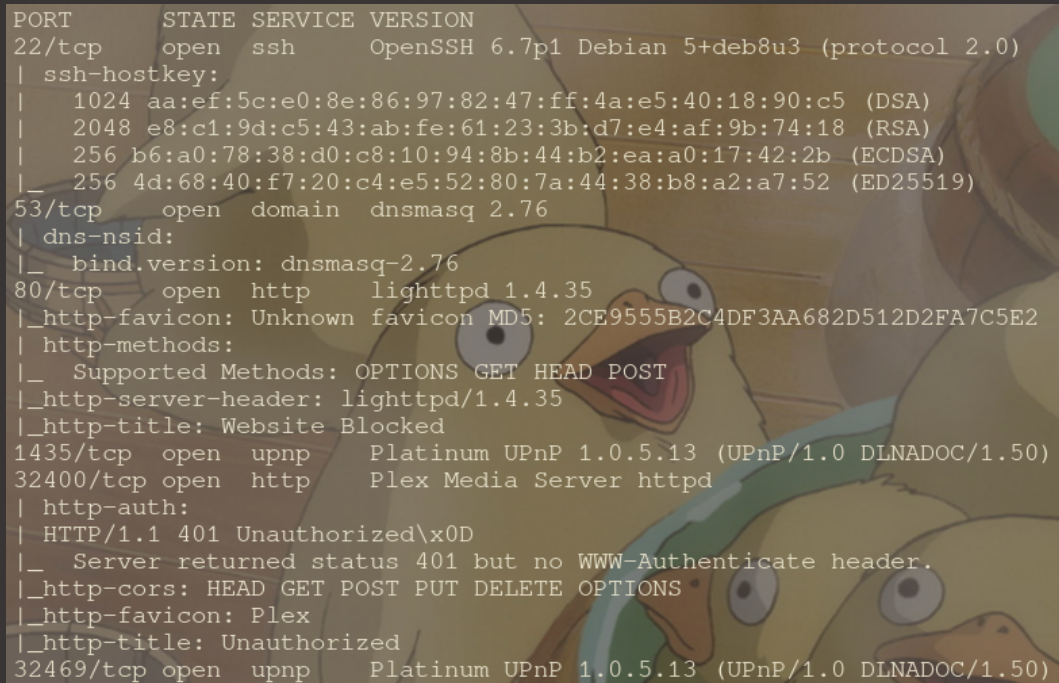
Before we start the party though, there's some cursory information that we should know before attempting this challenge. Mirai (未来) means future in Japanese (which is pretty interesting when you consider the intent of the developers). The Mirai botnet initiated a nice little scuffle between some teenagers and the internet's IoT devices. Primarily, the malware would infect Linux IoT systems and try to login with the default password. If there was success in logging in, the malware would use those devices to search for other vulnerable public-facing devices and take care of menial DDoS work. The group then used this network to pressure companies to pay for "protection". We may draw a few conclusions from this information:

1. The box is most likely a Linux IoT device
2. We should try the default credentials on telnet or ssh

Without further ado, let's get into it!

2 Walkthrough

2.1 Scanning



```
PORT      STATE SERVICE VERSION
22/tcp    open  ssh      OpenSSH 6.7p1 Debian 5+deb8u3 (protocol 2.0)
|_ ssh-hostkey:
|_ 1024 aa:ef:5c:e0:8e:86:97:82:47:ff:4a:e5:40:18:90:c5 (DSA)
|_ 2048 e8:c1:9d:c5:43:ab:fe:61:23:3b:d7:e4:af:9b:74:18 (RSA)
|_ 256 b6:a0:78:38:d0:c8:10:94:8b:44:b2:ea:a0:17:42:2b (ECDSA)
|_ 256 4d:68:40:f7:20:c4:e5:52:80:7a:44:38:b8:a2:a7:52 (ED25519)
53/tcp    open  domain   dnsmasq 2.76
|_ dns-nsid:
|_ bind.version: dnsmasq-2.76
80/tcp    open  http      lighttpd 1.4.35
|_ http-favicon: Unknown favicon MD5: 2CE9555B2C4DF3AA682D512D2FA7C5E2
|_ http-methods:
|_ Supported Methods: OPTIONS GET HEAD POST
|_ http-server-header: lighttpd/1.4.35
|_ http-title: Website Blocked
1435/tcp  open  upnp      Platinum UPnP 1.0.5.13 (UPnP/1.0 DLNADOC/1.50)
32400/tcp open  http      Plex Media Server httpd
|_ http-auth:
|_ HTTP/1.1 401 Unauthorized\x0D
|_ Server returned status 401 but no WWW-Authenticate header.
|_ http-cors: HEAD GET POST PUT DELETE OPTIONS
|_ http-favicon: Plex
|_ http-title: Unauthorized
32469/tcp open  upnp      Platinum UPnP 1.0.5.13 (UPnP/1.0 DLNADOC/1.50)
```

Nmap The very first scan is an nmap scan with augmented flags that fork from the usual options. The scan we'll be using is:

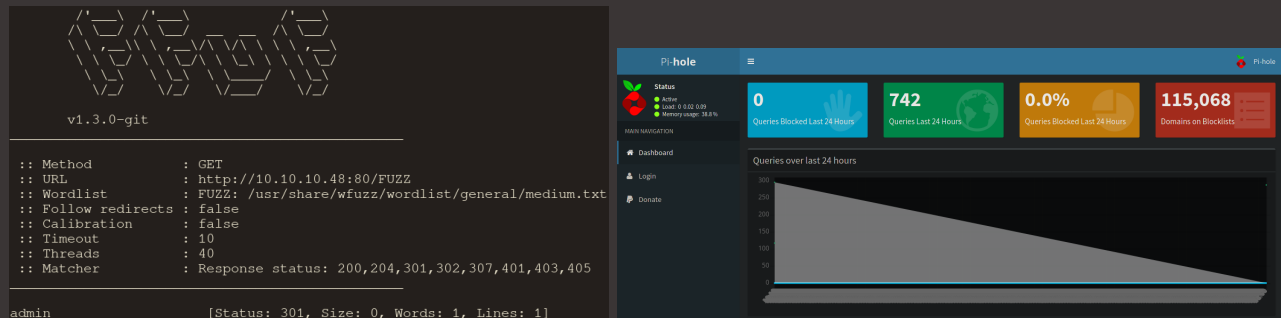
```
sudo nmap -p- -T4 -A -v 10.10.10.48 > mirai-scan
```

The results show open ports:

22	53	80
1453	32400	32469

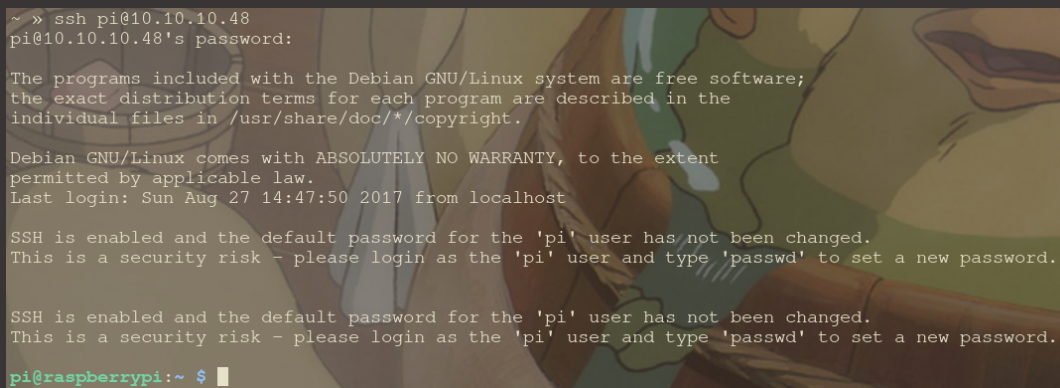
Right away we see that the default **ssh** and **http** ports are open, which is a normal starting place for challenges like this, as well as a few other non-standard ports such as a Plex media server and universal plug-n-play (another thing you don't want running). Don't shy away from doing a full port scan or changing the options if not enough information is gleaned from the initial scan like we did in this instance; it's better to be safe with full knowledge rather than continuing blind. In this specific scenario, we had to include the verbosity flag and change the timing to get a workable amount of information.

Web Fuzzing We'll want to add `10.10.10.48 mirai.htb` to your `/etc/hosts` file. We'll use `ffuf` and `dirbuster` to look for interesting stuff here with `wfuzz`'s medium wordlist. We find an `/admin` directory and see that the server is running Pi Hole, a neat network-wide firewall utility. With this information, we have more than enough to gain access to the system.



2.2 Gaining Access

Getting the foothold If you've ever set up a Raspberry Pi before, you'll know that Raspbian OS comes with a set of default credentials: `pi:raspberrypi` which can be used to access the machine through `ssh` and set it up. Unfortunately, some people neglect to change the password or delete the user altogether after the setup process.



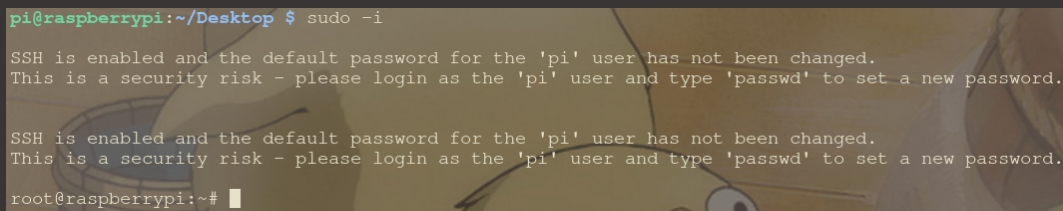
2.3 Maintaining Access

In some cases, the host will have an IPS or IDS solution set up to catch attackers that just accessed the system; sometimes there are even cron jobs to close non-enabled services or close connections to keep things secure. In these cases, we'd have to set up a script to counter the cron job and reopen the connection after a certain amount of time or enact evasion techniques. We got super lucky this time and only have to note the default (and quite easily memorizable) credentials. You should always store these just in case, they'll come in handy later when you're making writeups or trying to remember that one thing you did when working on other machines. You should take notes and keep screenshots as well to immortalize your achievements. You can also jot down some diagrams such as network layouts or mind-maps which will help keep you on track; you can use paper and a pen and grab a picture with your phone if you're feeling extra creative.

2.4 Escalating Privileges

Now that we're settled in with the `pi` user and have the user hash, we can now try to grab a root shell. There are a few possibilities we can pursue to escalate privileges and some require a few more steps:

- Using the `sudo -i` command
- Check for world-writeable files
- Perform lateral escalation to another user and repeat the list



```
pi@raspberrypi:~/Desktop $ sudo -i
SSH is enabled and the default password for the 'pi' user has not been changed.
This is a security risk - please login as the 'pi' user and type 'passwd' to set a new password.

SSH is enabled and the default password for the 'pi' user has not been changed.
This is a security risk - please login as the 'pi' user and type 'passwd' to set a new password.

root@raspberrypi:~#
```

We'll start with the `sudo -i` command since it's the easiest command to remember and it will give us the most amount of useful information to start with. Sometimes, we'll be lucky enough to get a list of commands that we can run with sudo permissions; in this instance we get pushed straight into a root shell, which is pretty darn cool. For the sake of education, we'll go over the other steps in the process. To check for world-writeable files, we can issue the command `find / ! -path "/proc/*" -perm -2 -type f -print 2>/dev/null` to search through the entire system (except for any `proc` directories) for files that we can modify and throw all those pesky errors into `/dev/null` where they belong. These few processes in the loop is a vast oversimplification of the enumeration you want to do on the system. You'll usually want to collect:

- Running processes
- Cron jobs
- Open network services (open to the world and to localhost)
- Users on the system
- Other system information that may stand out

Just add those into the loop where necessary and the rest should fall into place while you document the commands you run with the results. The most common way to escalate privileges laterally is to check for a database service such as `MySQL` or `NoSQL`, or to look at configuration files for running network services like `Apache` or `Nginx`. A good portion of the time, we can get some credentials for another user and repeat the process of enumerating the system from the standpoint of the new user.

```

root@raspberrypi:~# lsblk
NAME        MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
└─sda1       8:1    0   1.3G  0 part /lib/live/mount/persistence/sda1
└─sda2       8:2    0   8.7G  0 part /lib/live/mount/persistence/sda2
sdb          8:16    0    10M  0 disk /media/usbstick
sr0         11:0    1  1024M  0 rom
loop0        7:0    0    1.2G  1 loop /lib/live/mount/rootfs/filesystem.squashfs
root@raspberrypi:~#

```

Getting back on track, since we were able to get a root shell quickly, we can check around for the `root.txt` file, which is usually in the home directory for the root user. In this scenario, however, we are thrown a curveball in the form of a message pointing to a usb drive on the system. To get the required information, we can just use `lsblk` to look at the block devices and see where things are mounted. In our case, the usb drive is mounted to `/dev/sdb` and can be accessed in the `lstinline/media` directory under the same name shown in the `lsblk` output. When we `cd` to that directory, we see a few files which aren't very comforting. If we `cat` the `damnit.txt` file, we see that the root hash was deleted from the usb drive. Time to get into the weeds (kind of).

There are a few methods to check for deleted files:

- find an open file with `lsuf`
- make an image of the drive
- use `hexdump` on the drive
- use `strings` on the drive

```

root@raspberrypi:/media/usbstick# cat damnit.txt
Damnit! Sorry man I accidentally deleted your files off the USB stick.
Do you know if there is any way to get them back?

-James

```

Sometimes, a running process will grab data from a file, like an environment file or a config file. Maybe we can get away with finding a running process that opened the `root.txt` file by issuing the `lsuf | grep "root"` command. Unfortunately nothing came up, but if we got a process back, we could look into the `/proc` directory to see if we could find some saucy nuggets. The `/proc` directory is Linux's process pseudo directory that houses information like a file descriptor subdirectory which contains links to all the still-open files used by the process in question. The path would look something like `"/proc/process_id/fd/file_descriptor"`. In this case, we'll move on to making an image of the device with `dd` or `dcfldd` (the more enhanced 1337 version of `dd` created by the Department of Defense Computer Forensics Lab; seriously, check it out). In both cases, we want to specify the `/dev/sdb` block as input with the `if` option and set the `of` (output file) to the home directory of the pi user or a temporary file which we'll download on our own device (or not). We can use `testdisk`, `ddrescue`, or `foremost` to check out the discreet files that were on the usb disk before its demise. There seems to be a `root.txt` file in there when checking it out, but it shows 0 bytes. What the heck? At least we know the file is in there somewhere. Let's move onto other simple methods of grabbing lost data. `Hexdump` can be used on the image we took of the usb drive to check the contents in hex format; I'd highly recommend using the `C` and `L` flags to save yourself a great deal of time and a headache. Like we saw in `testdisk`, `root.txt` is near the bottom of the file list, so we'll start from there and work our way up. We have the root hash, neat! But let's blow our socks off by cheating a bit. Run the `strings` command on `/dev/sdb` to see all strings in that data file. `Strings` grabs all text data from a given file or path and returns it as output. In this scenario, near the bottom, above the contents of the `damnit.txt` file, we see the hash clearly and ready to be submitted. Congratulations on rooting the machine, but our job's not over yet.

```
root@raspberrypi:/media/usbstick# strings /dev/sdb
>r &
/media/usbstick
lost+found
root.txt
damnit.txt
>r &
>r &
/media/usbstick
lost+found
root.txt
damnit.txt
>r &
/media/usbstick
2]8^
lost+found
root.txt
damnit.txt
>r &
3d3e483143ff12ec505d026fa13e020b
Damnit! Sorry man I accidentally deleted your files off the USB stick.
Do you know if there is any way to get them back?
~James
0080a800 33 64 33 65 34 38 33 31 34 33 66 66 31 32 65 63 |3d3e483143ff12ec|
0080a810 35 30 35 64 30 32 36 66 61 31 33 65 30 32 30 62 |505d026fa13e020b|
0080a820 0a 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
0080a830 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
*
0080ac00 44 61 6d 6e 69 74 21 20 53 6f 72 72 79 20 6d 61 |Damnit! Sorry mal|
0080ac10 6e 20 49 20 61 63 63 69 64 65 6e 74 61 6c 6c 79 |n I accidentally|
0080ac20 20 64 65 6c 65 74 65 64 20 79 6f 75 72 20 66 69 | deleted your fi|
0080ac30 6c 65 73 20 6f 66 66 20 74 68 65 20 55 53 42 20 |les off the USB |
0080ac40 73 74 69 63 6b 2e 0a 44 6f 20 79 6f 75 20 6b 6e |stick..Do you kn|
0080ac50 6f 77 20 69 66 20 74 68 65 72 65 20 69 73 20 61 |ow if there is a|
0080ac60 6e 79 20 77 61 79 20 74 6f 20 67 65 74 20 74 68 |ny way to get th|
0080ac70 65 6d 20 62 61 63 6b 3f 0a 0a 2d 4a 61 6d 65 73 |em back?..-James|
0080ac80 0a 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
0080ac90 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
*
00a00000
```

2.5 Covering Tracks

Any artifacts created on the system should be deleted first to get rid of the most glaring activities completed on the system. Let's retrace our steps and clean them up a bit so that we don't leave such a large footprint on the machine. Firstly, if you decided to put the recovery image into a temp directory or into the pi user's directory, feel free to get rid of that first. This is akin to cleaning up the big chunks of glass before taking out the mop and broom to clean up the little bits and spilt milk. After getting the created artifacts off the system, let's grab the broom and dustpan. As root, we can modify the log files of the pi user to make it like we never accessed that account; most of the stuff we'll be working on can be found in the `/var/log` directory. After all user stuff has been swept from the logs, including any command history, we'll get the mop and finish cleaning up the spilled milk. In some cases, the `bash_history` and other command history files are symlinked directly to `/dev/null`, which is basically a black hole for output. This helps us in covering our tracks, but it also can be annoying if we're feeling cheeky or just want to see how others completed certain tasks after we've rooted the box. Delete entries from the logs from the time you connected and started running commands as root as well. Be careful to not clear out the entire log file, as that looks super suspicious and will definitely flag the spider sense of the sys admin (most of this would anyway, but there's a chance that unless the specific changes are logged, we can get away with making the sys admin think some alerts were false positives). Finally, clear access logs for root and gtf0. We're done; now you can rest easy and move onto the next one!

3 Conclusion

3.1 Final thoughts

For this particular writeup, I've left the mitigation of vulnerabilities in the pseudo report found on the next page; this just clears everything we talked about here, and rounds out this writeup for the aspiring blue-teams that want to get a head-start on getting rid of pesky bugs commonly found on systems. Overall, the challenge was fun for me personally; I learned a lot about data recovery and what tools could be useful in troubleshooting usb drives. I haven't used strings or hexdump outside of reverse engineering in a while, so that was a refreshing change of pace. Like I stated in the introduction, I try and create these writeups in Spanish and Japanese as well to reach a wider audience and help those getting into cybersecurity which may not have access to training materials in their native language, or can't take advantage of existing training materials due to the language barrier. Have a great day, and thank you for spending this time learning with me. Happy hacking!

3.2 Tracking & Reporting

Customer Arrexel

Documents A scope of work and non-disclosure agreement are needed for this engagement (not included, use your imagination)

Engagement

- Type - Black Box
- Scope - Remote, WebApp, App
- Timeline:
 - From: Feb 10, 2021
 - To: Feb 11, 2021
 - Status Updates: None

Infrastructure Debian Linux

Test Accounts N/A

Objectives

- Obtain user access on the system
- Exfiltrate a user hash
- Obtain root access on the system
- Exfiltrate a root hash

Tasks

- Conduct a network scan on the host
- Gain a foothold on the system
- Complete the first two primary objectives
- Enumerate the system from the PoV of the user
- Escalate privileges
- Complete last two primary objectives
- Cover tracks on system
- Create documentation of the engagement

Tools

nmap	ffuf	ssh	cat	dd	dcfldd	testdisk
ddrescue	foremost	scp	strings	hexdump	lsof	find

Findings From the beginning of the engagement, I found that there were public-facing non-essential network services running on the machine in scope. After finding the type of service running on the default http port, I attempted to ssh into the machine using default credentials and succeeded in gaining a foothold onto the system. User data was unencrypted. Upon attempting to find commands with sudo privileges, I was immediately escalated to a root shell on the target system. I found a non-essential usb drive mounted onto the filesystem. Unencrypted data was present on the usb drive, and I was able to exfiltrate data that was supposed to be inaccessible.

Recommendations

- Shut down and disable all non-essential public-facing network services (especially uPnP and Plex media server)
- Change the default user password or remove the user entirely after initial configuration
- Encrypt all sensitive user information
- Disallow sudo permissions to basic users
- Unmount usb drives when not in use
- When deleting contents of a drive, complete a bit-level overwrite as well to mitigate the possibility of recovering sensitive data