



**Proof of concept; a worm botnet.**

**Abigail ECCLESTON, B00140442**

**Jérémy PRIMARD, B00168148**

**Tytus KOPERA, B00140074**

**School of Informatics and Cybersecurity, Faculty of Computing, Digital & Data,  
Technological University Dublin**

**Submitted to Technological University Dublin in partial fulfillment of the requirements for  
the degree of**

***Bachelor of Science in Computing in Digital Forensics & Cyber Security***

**Supervisor:**

**Mark Lane**

**May 2024**



## Declaration

I hereby certify that this material, which I now submit for assessment on the programme of study leading to the award of Degree of **Bachelor of Science in Computing in Digital Forensics & Cyber Security** in Technological University Dublin, is entirely my own work except where otherwise stated, and has not been submitted for assessment for an academic purpose at this or any other academic institution other than in partial fulfilment of the requirements of that stated above.

Dated: 04/15/2023

Abigail ECCLESTON

Jérémy PRIMARD

Tytus KOPERA

## Abstract

---

### *Abstract*

Whatever one host can do by itself, one thousand hosts can do it better together. Malicious actors spread their networks of zombie hosts to grant them the power of many machines. They pose a challenge to law enforcement, companies and targeted individuals given their power to cause damage as well as resilience from being disrupted.

We decided to learn about the intricacies of this threat by creating our own botnet, written in Rust. Our idea was to use worm's self replicate ability to conceal the hacker's host ip address, so that a bot know only the ip address of the host that have infected it, this way, it can be a lot more difficult for ISP to take down the botnet. Rust is known for its robust type system, safety and concurrency, as well as its fantastic performance making it a fantastic language to used for this botnet. The aim was to delve into the low level world of programming networking functionality, hence why Rust was used. We used the Rust library documentation and Rust's compiler to learn the language along the way and getting acquainted with how a networking program should be written in a low level. This botnet utilizes the ShellShock ie. CVE-2014-6271 which exploits a flaw in Bash which if it is couple with a webapp that needs to enter system command, allows a malicious actor to remotely execute arbitrary commands on the vulnerable host. Communication is encrypted with TLS thanks to OpenSSL and Rustls. Because of our needs for the demo, we made a primitive ip targeting system that allow only 1 to 1 parent-child node relations.

Despite the age of the exploit, it is very likely that it stays some website using this specific outdated bash version, meaning that our malware could potentially make some bots if release in nature. The exploit itself wasn't really important because the only thing that was important to us, was to be able to exploit by calling a bash script, so that it can be easier to make our botnet run on different exploit. A possibility of future improvment would be to make it run on multiples exploits at the same time. Therefore, it is nessessary to have different botnets to use, in order to implement protection against botnet in IPS and IDS.

*Keywords:* Botnet, malware, worm, rust, C2, shellshock, TLS.

## Contents

---

<b>Abstract</b>	<b>1</b>
<b>I Introduction</b>	<b>3</b>
I.1 Definitions . . . . .	3
I.2 Origins of the project . . . . .	3
I.3 Preliminary research on existing botnets and an existing worms that appears in the last 25 years. . . . .	4
<b>II Our implementation and design</b>	<b>5</b>
II.1 Things necessary for the project we had to either find or makes. . . . .	5
II.2 Botnet design . . . . .	6
II.3 Lab setup . . . . .	6
II.4 Botnet communication . . . . .	7
<b>III Part 2</b>	<b>9</b>
III.1 Why Rust? . . . . .	9
III.2 Flow of the program . . . . .	9
III.3 Client - A closer look . . . . .	9
III.4 Server - A closer look . . . . .	9
<b>IV Part 3</b>	<b>11</b>
IV.1 Something part . . . . .	11
IV.2 Something part . . . . .	11
IV.3 Something part . . . . .	11
<b>Conclusion</b>	<b>12</b>
<b>Glossary</b>	<b>13</b>
<b>References</b>	<b>14</b>

# **I. Introduction**

---

## *I.1. Definitions*

The world is relying on computers more with each passing year. It's necessary more than ever to know how a computer virus functions. One of the many types of malwares is the worm. A worm is a self-replicating type of malware, which spreads out and takes more computers, these are called victims. Its use for a black hat hacker is usually to launch ransomware and other types of denial-of-service attacks on victims. One of the many dangerous features of a worm is how it does not need a central controlling component to propagate. A central controlling component is where the worm can be controlled and given instructions, this is also known as its "head". A worm virus can also conceal the party of which created or distributed the malicious software.

Botnets are another type of computer virus, which hijacks and takes control of multiple machines to violate laws or regulations. This type of malware could be instructed to launch a cyber-attack, such as Distributed Denial of Services (DDOS) attacks, mine crypto money or crack passwords. A botnet need a machine to give it's orders, without it, there would be no orders and no attacks, so no botnets.

## *I.2. Origins of the project*

As a result of the botnet virus's inability to continue propagating without its head, our project idea began. We asked ourselves what if we took aspects from the two viruses and combined them. Specifically, the worm's ability to conceal the preoperatory and the botnets capability to control the bots of itself. Moreover, our idea and concept is to make a botnet that spreads like a worm and conceals the address of the 'head' to the majority of the infected machine. The address being its IP address, an IP address is an unique code that identifies a computer or a certain computer network. This being a monumental task within our ability and timeframe we simplified our idea to being proof of concept, a worm botnet. We reiterate that our project is solely proof of concept and is intended to only be showcased in our project demonstration. We recognize that our project would need immense improvements in several areas to overcome its limitations. Again, we reiterate that this project isn't supposed to show the virus of the future. Its purpose is to show proof of concept and demonstrate that there isn't enough research on a worm botnet. In fact, the closest documentation we could find that was congruent with our concept was peer to peer botnets. Which has similar aspects to what we want to achieve, but not precisely. (<https://ieeexplore.ieee.org/abstract/document/5684002/>) (<https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=5de6987574e48076fa5f024f347d44c77a6fa080>) We where also really interested to know how a botnet, and how a worm would works, and this project gave us the knowledge we always dreamed to get.

### *I.3. Preliminary research on existing botnets and an existing worms that appears in the last 25 years.*

Before we delve into the implementation aspect of our botnet let's first examine well-known worms and botnets.

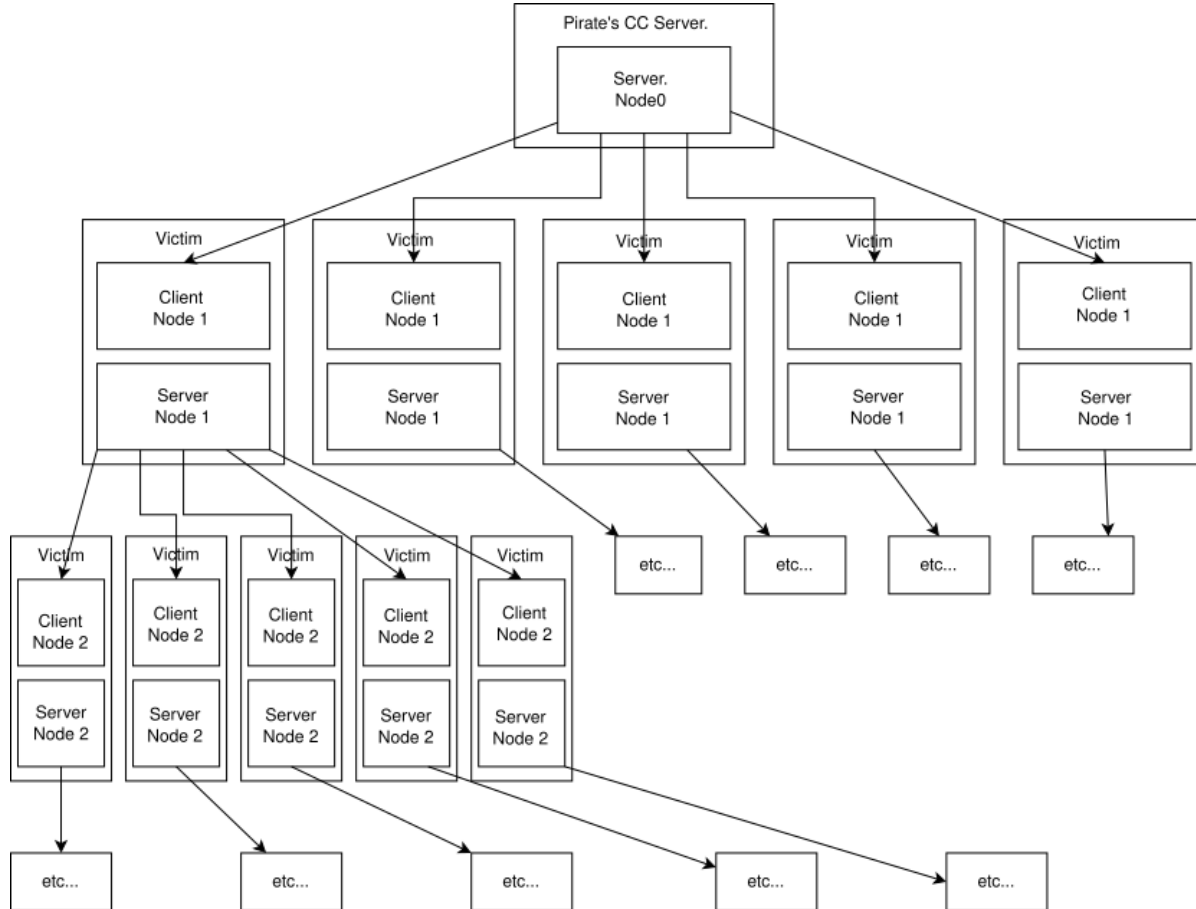
Worm virus WannaCry ([https://en.wikipedia.org/wiki/WannaCry\\_ransomware\\_attack](https://en.wikipedia.org/wiki/WannaCry_ransomware_attack)) is a bot which first appeared in 2017. This bot exclusively targeted Microsoft Windows Software. It spread itself exploiting a critical vulnerability of the Server Message Block (SMB), this being on every Windows machine it was extremely serious. The way this worm works is standard in the realm of worm viruses. The worm will attack if the machine is vulnerable, it will drop the exec file. This file's goal is to self-replicate the worm, install ransomware and target other machines (<https://www.csoonline.com/article/563017/wannacry-explained-a-perfect-ransomware-storm.html>)

Another example is the opensource Mirai botnet([https://en.wikipedia.org/wiki/Mirai\\_\(malware\)](https://en.wikipedia.org/wiki/Mirai_(malware))) It's a botnet that spreads over ports 23 and 2323, in order to make new bots. The way it works is simple, it tries to brute force the password of an IoT device. It achieves this by using default built-in credentials, for example default usernames, passwords etc. However, this relies on the fact that it hasn't been changed. Once this is complete the bot awaits its instructions. Usually, the bot is instructed to launch the cyber-attack Denial of Service (DOS), which is an attack that floods the server or network with too much traffic, causing it not to function properly. This cyber-attack is launched against a DNS server or a big cloud company.

Botnet Hajime is another example ([https://en.wikipedia.org/wiki/Hajime\\_\(malware\)](https://en.wikipedia.org/wiki/Hajime_(malware))). This botnet's simply goal was to protect devices against the previously talked about Mirai botnet, by closing telnet ports. Hajime is a well-known botnet it uses peer to peer capabilities; it does this to hide its perpetrator address. Thus, makes it far more difficult to be blocked by ISP to take down the botnet ([https://thehackernews.com/2017/04/vigilante-hacker-iot-botnet\\_26.html](https://thehackernews.com/2017/04/vigilante-hacker-iot-botnet_26.html))

## II. Our implementation and design

---



*II.1. Things necessary for the project we had to either find or make.*

Our project had different parts.

- Making a sandbox environment while ensuring the worm does not escape We used qemu and unshare to make an environment that couldn't reach the external network. In the end it was unnecessary because we decided that for the demonstration the victim ip will be known at launch time, to prevent unexpected results in the final demo.
- Finding an exploit to use We needed the exploit to perform remote code execution, be easy to setup and use because the point of this project was not to create an intricate exploit, but rather to build a botnet. We spent hours looking for the perfect exploit and in the end we found a repository on github which performed in a way that we wanted. (<https://github.com/opsxcq/exploit-CVE-2014-6271>) Thanks to the work of opsxcq, we had an easy vulnerability to exploit. This vulnerability was CVE-2014-6271, known as Shellshock. In opsxcq's repository there was a docker image vulnerable to Shellshock which was run by a simple command.

There were two other vulnerabilities in his github that we were interested in but ultimately chose to not use: CVE-2016-10033 and CVE-2017-7494. There were both unsuitable for our application and would have required too much effort to get working. We also considered log4j but decided to use an easier exploit to implement.

- Demonstrate the botnet In order to demonstrate that the botnet worked we had to prove that the botnet has really taken control of several machine. To accomplish this we created a VM called vulnerableDOS. This VM would run Wireshark and, once the botnet has taken control of the bots, receive and display the pings in the interface of Wireshark. At first we wanted to make our botnet run an actual DDOS attack on the vulnerableDOS machine, but the purpose of the project was not to make it run a complex attack because our goal was just to prove that we have indeed taken control of the vulnerable machines. Because of this, we chose to simply ping the vulnerable DOS VM.

Of course, we also had to make the botnet itself

## *II.2. Botnet design*

Designing and making the botnet was a real challenge. The architecture of our botnet would consist of a server and a client. The server had three tasks. Firstly it runs the exploit against the victim. Secondly, it waits for a response from the victim to run a Command and Control server (CC), to deliver its order. The first order will always be to replicate the server on the victim, and the second one to ping the vulnerableDOS VM. Thirdly, the server runs a file sharing server (fs), so that every time the CC server asks the victim to download a file, the file can be downloaded.

On the CC and the fs server, we implemented a TLS connection, using rustls. The certificates used are self signed certificates, and every time the server is copied into the victim, openssl is used to generate the certificates. We were really lucky to find that openssl 1.0f was installed on the vulnerable image. If the docker image was made two years earlier, openssl probably wouldn't have been installed, forcing us to abandon the idea of using encrypted connections.

The goal of the client is to connect to the server, get its orders, and execute whatever the orders entail.

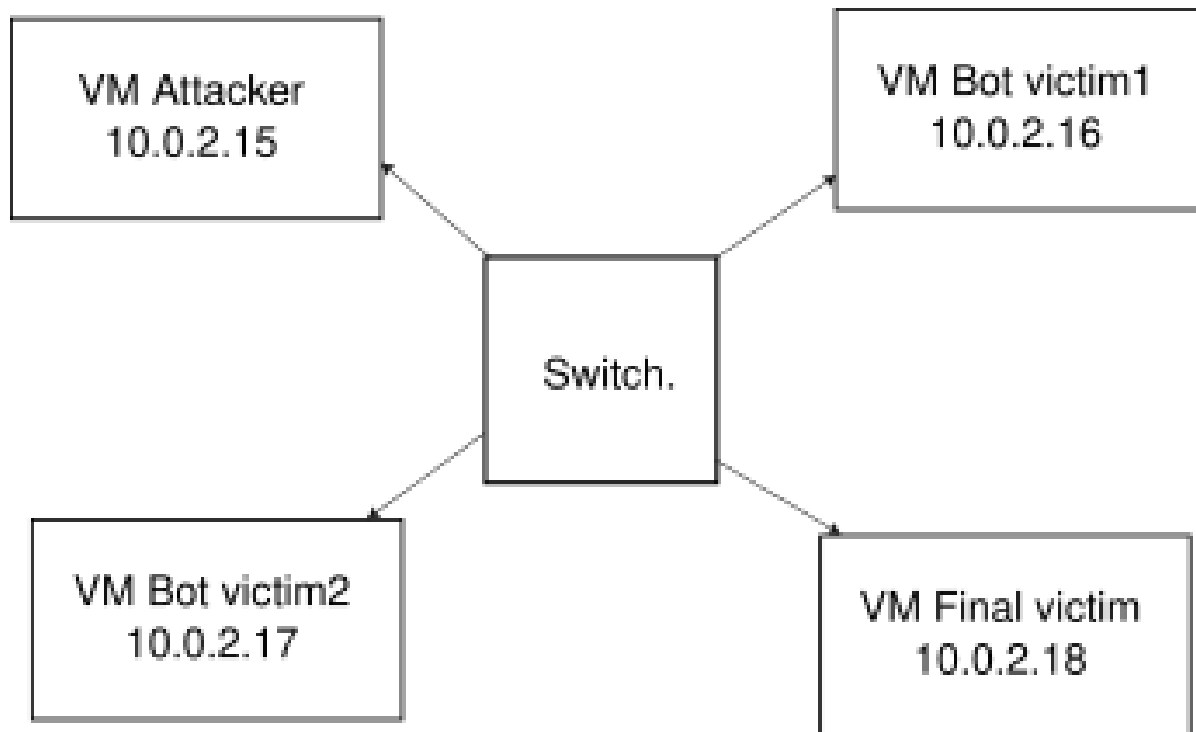
The server reads TLS certificates from the conf folder, and the orders file that it transmits to the victim. It also creates files in the www directory eg. the list of IPs that the next victim has to attack, a config file to easily generate new certificates, the exploit to run in order to run the client on the victim, and an installation script. In summary, the server permits the client to install whatever the servers serves and to replicate the server itself.

## *II.3. Lab setup*

To set up the lab environment we first made a VM, with the dinit init system on it, as well as basic packages. We did this to be aware of all the programs running on the VM so that we didn't lose excessive time searching for bugs caused by clashes in the OS. We then made a VM to be the head of our botnet, 2 vulnerable VMs and 1 more VM named vulnerableDOS. We called the first VM attacker, as it was going to be the initial aggressor in the demonstration. The two vulnerable VMs were called vulnerable1 and vulnerable2. These were going to be infected by the malware and



added to the botnet. vulnerableDOS was going to be the victim of the botnet's ping attack.



This means that our botnet project can be considered a success if we manage to make vulnerable1 and vulnerable2 ping vulnerableDOS.

#### *II.4. Botnet communication*

The following is the flow of the botnet's activity: Step one. The server finds an potential target, within the demo, this target is specified. It runs a BASH script, called exploit.sh, which executes the exploit against the target. The exploit controls the victim in order to make the victim download and execute the client binary. In our case, the client is installed and run in '/tmp/botnet/'.

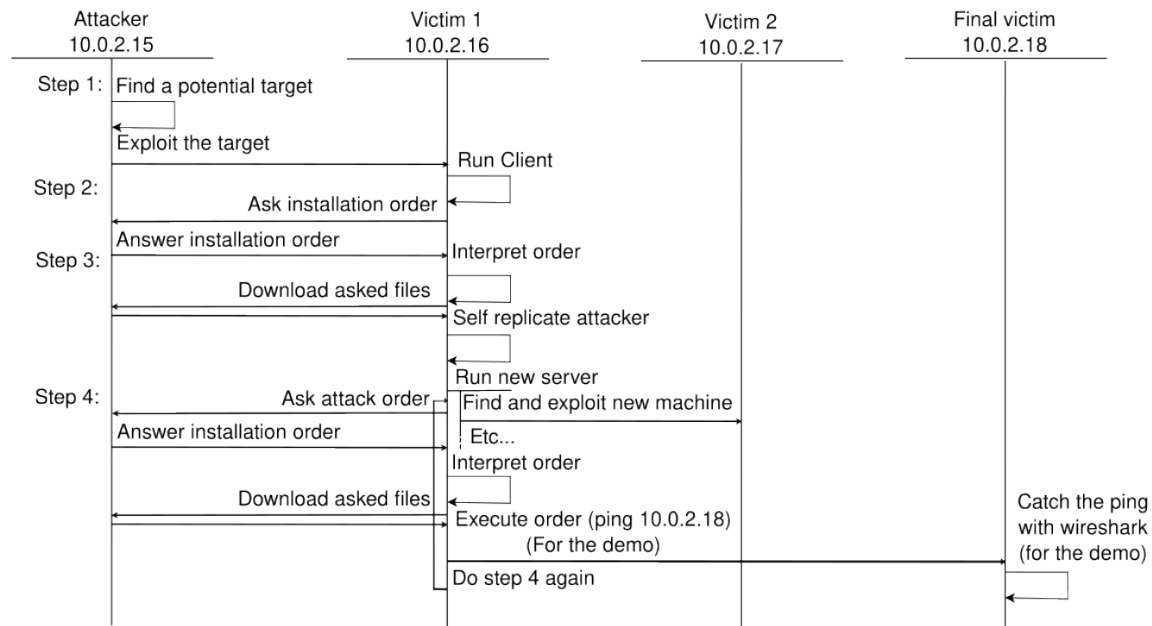
Step two. The client requests the server for its first instruction, called order1. The server answers with the file and the client saves it in '/tmp/botnet/conf/'.

Step three. The client reads the received order and every time it reaches a line beginning with 'download', it requests the filesharing server for the respective file to download which is saved to '/tmp/botnet/downloaded/'. Everytime the client reaches a line beginning with 'execute', it executes or runs the file.

Normally, only the file 'installation.sh' is executed at this stage. This script copies all the downloaded files from '/tmp/botnet/downloaded' to their respective folders. Now the server is run on the victim, which begins to look for other victims.

Step four. Finally, once the victim runs its version of the server, it asks for its attack order from the CC server, which is called order2. Typically, an order2 file will ask the client to download and execute a BASH script. This BASH script will run the attack. In the case of the demo, it will only

run 'ping 10.0.2.18' for simplicity.



Note that every time it receives an order, it will save it, and make it accessible to the server so that it can relay the orders to its child node, ie. the victim of the server that is being run. The same is true for every file that is downloaded.

Having explained the thought process behind the design of the project, let us now look at the code.

## III. Part 2

---

### *III.1. Why Rust?*

Rust is a multi-paradigm low level programming language which emphasizes memory safety, strict types and high performance. Despite its novel features (and associated learning curve), such as variable ownership and its omission of a null variable, it is already a loved language among developers, which forces the developer to write safer and more performant code. Despite it lacking as many footguns as C/C++, it allows a great deal of control over what happens in a program. Though young, Rust is seen as the spearhead for the languages of the future, looking to replace C++ as the dominant low level language. Rust has been added to the Linux kernel, currently the only language to accompany C in this domain, and is being picked up by even the largest companies, such as Google. On the opposite side of the spectrum, hackers have begun writing in Rust, due to its aforementioned strengths and the current inability for malware scanners to detect threats in the binary[reference]. All of these reasons combined form our argument for learning Rust, which is what we had to do for this project.

### *III.2. Flow of the program*

### *III.3. Client - A closer look*

The main function of the main function for the client is to connect to the parent node's server. Firstly we call `set_working_directory` from the module `set_dir` to launch the program at the root of the source code, so that no pathing issues may arise. Ip address and port numbers are specified with the `args` functions, which can take command line arguments or read directly from a file. These slices are parsed as four unsigned 8 bit integers and two unsigned 16 bit integers and fed into `Ipv4Addr-new`. Next, the directories "www", "conf" and "downloaded" are created, so that the files that are downloaded from the server and those that are created by the client are in their respective directories. The `u64 number_of_order` is initialized to zero so that the client may keep track of connections to the server, which will be used to coordinate which files are to be downloaded (verify). The meat of the functionality exists inside the loop, where the host continuously establishes TLS connections to the C2 and file sharing servers. The first connection is established with the C2 server on port 7878. Then the `flow` function is called from the module `connection`. This allows data transfer and the receiving of files from the file sharing server in a safe and managed way. After this, the `number_of_order` variable gets iterated by one and the program sleeps for 5 seconds, before more data is received.

### *III.4. Server - A closer look*

Most of the code exists in the server part of the malware, and with good reason, this is the most important part of the source code. It dictates which files get dealt with in which way. Mainly we look to create and manipulate files for configuration and managing client connections.

Firstly we set a working directory to prevent pathing issues. Rustls is set up by calling `set_tls`, which reads the `certs.pem` and `privkey.pem` files in the `conf` directory and the returning `ServerConfig` struct is passed into the value `acceptor`. The ip address of the server is recorded and added to a file, which is truncated and located in the `www` directory. We reuse the ip address multiple times so having a copy of it in its own file is important. We clone the variable that is the ip address so as not to lose ownership of the variable and we use this copy as an argument to the function `hs_server` which is launched in its own thread. This function reads from the file `ip_victims` and checks if this address is found in another file, `used_ip`, which we create a vector for each value in the file. The reason for this is to not try to hack hosts which he either already attempted to hack. We remove this ip address from the vector and write to the file, truncating it. If there is a match then we do not run the exploit. If there is a match, we run the exploit with the arguments being the ip address of the server and the host to hack. The ip address of the victim gets pushed to the vector of already targeted ips and this vector gets written to `used_ip`. The ip variable gets cloned once more, as well as the `acceptor` variable, and are used as arguments for the function `file_share_server`, which is launched on a new thread. At this point, the victim should be successfully remotely controlled via the exploit and be attempting a connection to the filesharing and C2 server. The ports for the file sharing server and C2 server are 7870 and 7878 , respectively, by default. We chose these ports because they fall outside of the 0-1023 range, a range of port numbers where root is required to use them.

## IV. Part 3

---

### *IV.1. Something part*

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque viverra pulvinar dui ut venenatis. Suspendisse interdum ac tellus nec ultricies. Nullam eu bibendum ipsum. Pellentesque in ipsum vel orci ullamcorper malesuada in at turpis. Nulla facilisi. Quisque ullamcorper at sem eget porttitor. Ut sed mi fermentum, fermentum purus in, molestie sem. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque viverra pulvinar dui ut venenatis. Suspendisse interdum ac tellus nec ultricies. Nullam eu bibendum ipsum. Pellentesque in ipsum vel orci ullamcorper malesuada in at turpis. Nulla facilisi. Quisque ullamcorper at sem eget porttitor. Ut sed mi fermentum, fermentum purus in, molestie sem.

### *IV.2. Something part*

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque viverra pulvinar dui ut venenatis. Suspendisse interdum ac tellus nec ultricies. Nullam eu bibendum ipsum. Pellentesque in ipsum vel orci ullamcorper malesuada in at turpis. Nulla facilisi. Quisque ullamcorper at sem eget porttitor. Ut sed mi fermentum, fermentum purus in, molestie sem. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque viverra pulvinar dui ut venenatis. Suspendisse interdum ac tellus nec ultricies. Nullam eu bibendum ipsum. Pellentesque in ipsum vel orci ullamcorper malesuada in at turpis. Nulla facilisi. Quisque ullamcorper at sem eget porttitor. Ut sed mi fermentum, fermentum purus in, molestie sem.

### *IV.3. Something part*

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque viverra pulvinar dui ut venenatis. Suspendisse interdum ac tellus nec ultricies. Nullam eu bibendum ipsum. Pellentesque in ipsum vel orci ullamcorper malesuada in at turpis. Nulla facilisi. Quisque ullamcorper at sem eget porttitor. Ut sed mi fermentum, fermentum purus in, molestie sem. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque viverra pulvinar dui ut venenatis. Suspendisse interdum ac tellus nec ultricies. Nullam eu bibendum ipsum. Pellentesque in ipsum vel orci ullamcorper malesuada in at turpis. Nulla facilisi. Quisque ullamcorper at sem eget porttitor. Ut sed mi fermentum, fermentum purus in, molestie sem.

## Conclusion

---

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque viverra pulvinar dui ut venenatis. Suspendisse interdum ac tellus nec ultricies. Nullam eu bibendum ipsum. Pellentesque in ipsum vel orci ullamcorper malesuada in at turpis. Nulla facilisi. Quisque ullamcorper at sem eget porttitor. Ut sed mi fermentum, fermentum purus in, molestie sem. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque viverra pulvinar dui ut venenatis. Suspendisse interdum ac tellus nec ultricies. Nullam eu bibendum ipsum. Pellentesque in ipsum vel orci ullamcorper malesuada in at turpis. Nulla facilisi. Quisque ullamcorper at sem eget porttitor. Ut sed mi fermentum, fermentum purus in, molestie sem.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque viverra pulvinar dui ut venenatis. Suspendisse interdum ac tellus nec ultricies. Nullam eu bibendum ipsum. Pellentesque in ipsum vel orci ullamcorper malesuada in at turpis. Nulla facilisi. Quisque ullamcorper at sem eget porttitor. Ut sed mi fermentum, fermentum purus in, molestie sem. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque viverra pulvinar dui ut venenatis. Suspendisse interdum ac tellus nec ultricies. Nullam eu bibendum ipsum. Pellentesque in ipsum vel orci ullamcorper malesuada in at turpis. Nulla facilisi. Quisque ullamcorper at sem eget porttitor. Ut sed mi fermentum, fermentum purus in, molestie sem.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque viverra pulvinar dui ut venenatis. Suspendisse interdum ac tellus nec ultricies. Nullam eu bibendum ipsum. Pellentesque in ipsum vel orci ullamcorper malesuada in at turpis. Nulla facilisi. Quisque ullamcorper at sem eget porttitor. Ut sed mi fermentum, fermentum purus in, molestie sem. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque viverra pulvinar dui ut venenatis. Suspendisse interdum ac tellus nec ultricies. Nullam eu bibendum ipsum. Pellentesque in ipsum vel orci ullamcorper malesuada in at turpis. Nulla facilisi. Quisque ullamcorper at sem eget porttitor. Ut sed mi fermentum, fermentum purus in, molestie sem.

## Glossary

---

- **Botnet:** A type of malware which takes control of numerous machines to launch some attacks. Although it's usually used to launch denial of service attacks, it can also be used to crack passwords or even mine cryptocurrency.
- **Other example:** Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque viverra pulvinar dui ut venenatis.

## References

---

- [1] This website is the first example that came to my mind as an example.. <https://example.com>
- [2] This is another example to show peoples what is en entry in the references chapter.  
<https://secondexample.com>