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 herby 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 potentialy 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

I. Introduction

I.1. Definitions

In a world where more and more things rely on computing, it is necessarry to know how virus works, especially some of the worse ones. Worms, a type of virus that can self replicate in order infect more and more. It is usually use to lauch some ransomewares and other denial of service attacks. Because of it's self replicated nature, once launched, it doesn't need a head to continue spreading if it's codded to do that. It have also the good ability, depending what is it's purpose, to conceal the pirate. Botnets, another type of virus that takes control of multiples machines in order to do evil things, like Distributed Denial of Services (DDOS) attacks, or win more power in order to either mine cryptomoney, or crack password are another.

I.2. Origins of the project

This is where the idea of the project started. Because one of the huge flaw of botnet is that they need a head to works, we think of this idea. What if we can allied the ability to conceal the perpretrator of the attack from the worm, and the ability to control bots of a botnet? More precisally, our idea is to make a botnet that spreading like a worm, so that only a verry little quantity of machine actually knows the address of the head of the botnet. We remind here that our project is simply a proof of concept, and so, it's only meant to be shown in the final demo of the project. Several parts of it, if not all of them, would need real improvments, in order to overcome some of it's limits. Again, we remind that this project isen't meant to show the virus of the future, but only to show, and give a proof of concept, of a type of botnet there isen't enough research on it. Indeed, the closest thing we could find to our idea, was Peer to peer botnet, that's similar to what we want to do, but it's wasn't enough. (https://ieeexplore.ieee.org/abstract/ document/ 5684002/) (https://citeseerx.ist.psu.edu/ document?repid=rep1& type=pdf& doi=5de6987574e48076fa5f024f347d44c77a6fa080) We can also added to it that we were interested by the field

I.3. Preliminary research over an existing botnet and an existing worm that appears in the last 25 years.

Before begining to talk about the implementation of our botnet, let's find three examples of pretty well known worm and botnet, to get an idea of what already exist.

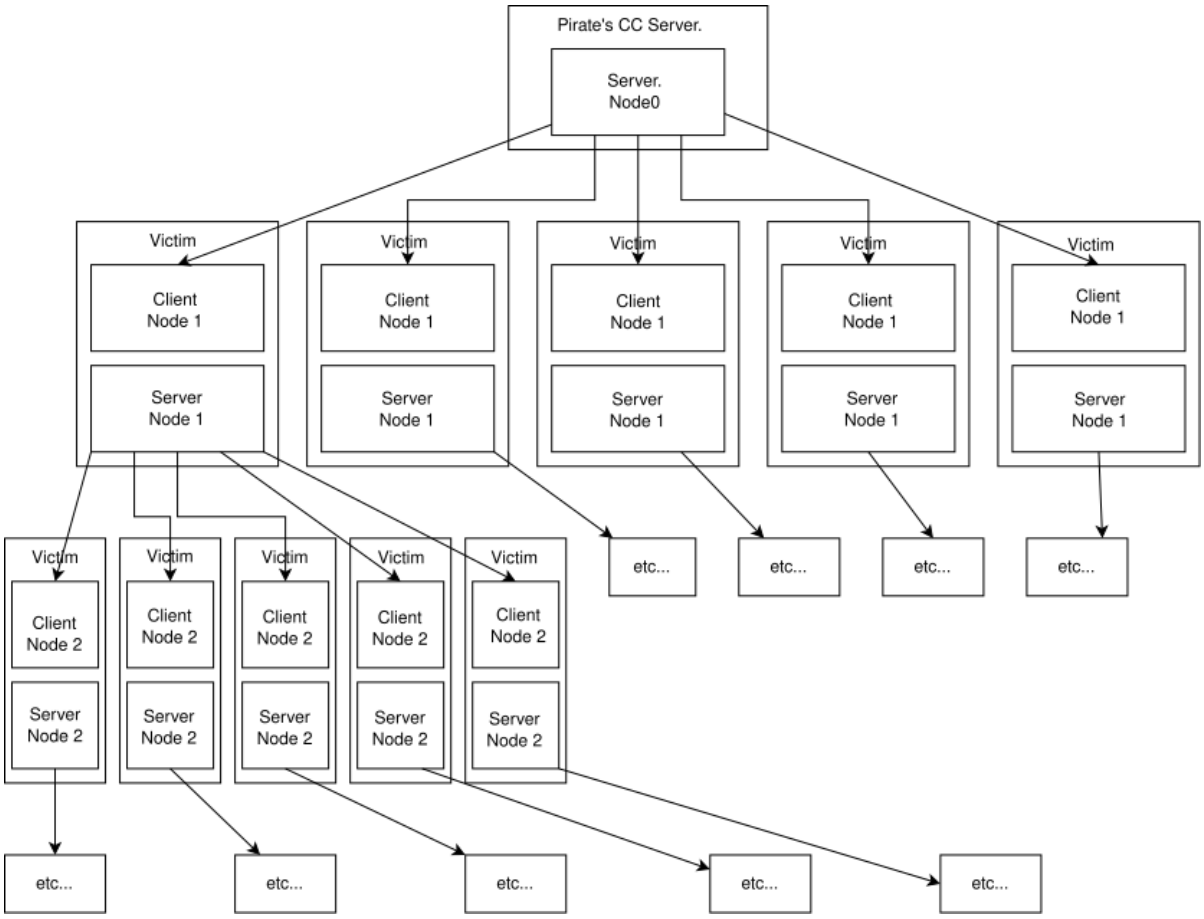
The first worm that we find was wanacry. (https://en.wikipedia.org/wiki/WannaCry_ransomware _attack) It's a bot which appeared on 2017 and targeted only Microsoft windows softwares. It was spread using a critical vulnerability of the SMB server that is on every windows machines. The way it's works is the same way as most worm, that means that if a machine is vulnerable, the worm will drop and exec a file, wich goal is to first self-replicate the worm, install a ransomware, and target other machines. (https://www.csoonline.com/article/563017/ wannacry-explained-a-perfect-ransomware-storm.html)

The other example we choosed, is the opensource, mirai botnet. (https://en.wikipedia.org/wiki/

Mirai_(malware)) It's a botnet spreading over ports 23 and 2323, in order to make new bots. It's way of working is simple. It tries to bruteforce some linux iot devices, using the default built-in credential, assuming that nobody changed it. After that, the bot awaits it's orders. Usually, it was to DOS a DNS server or a big cloud company.

Another botnet example, is the so called Hajime botnet (https://en.wikipedia.org/wiki/ Hajime_(malware)) which goal was simply to protect device against the mirai botnet, by closing telnets ports. This is a really well know botnet, using peer to peer capabilities, to hide it's owner address, and thanks to that, makes it a lot more difficult to be blocked by ISP to takes down the botnet. (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 makes.

Our project had different parts.

- Making a sandbox environment while ensuring the worm does not escape We used qemu and unshare to make an environnment 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 vulernable 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 to 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 accomlish 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 interace 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 vulnerables 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 it's 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 ask 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 year ealier, openssl probably wouldn't have been installed, forcing us to abandon the idea of using encrypted connections.

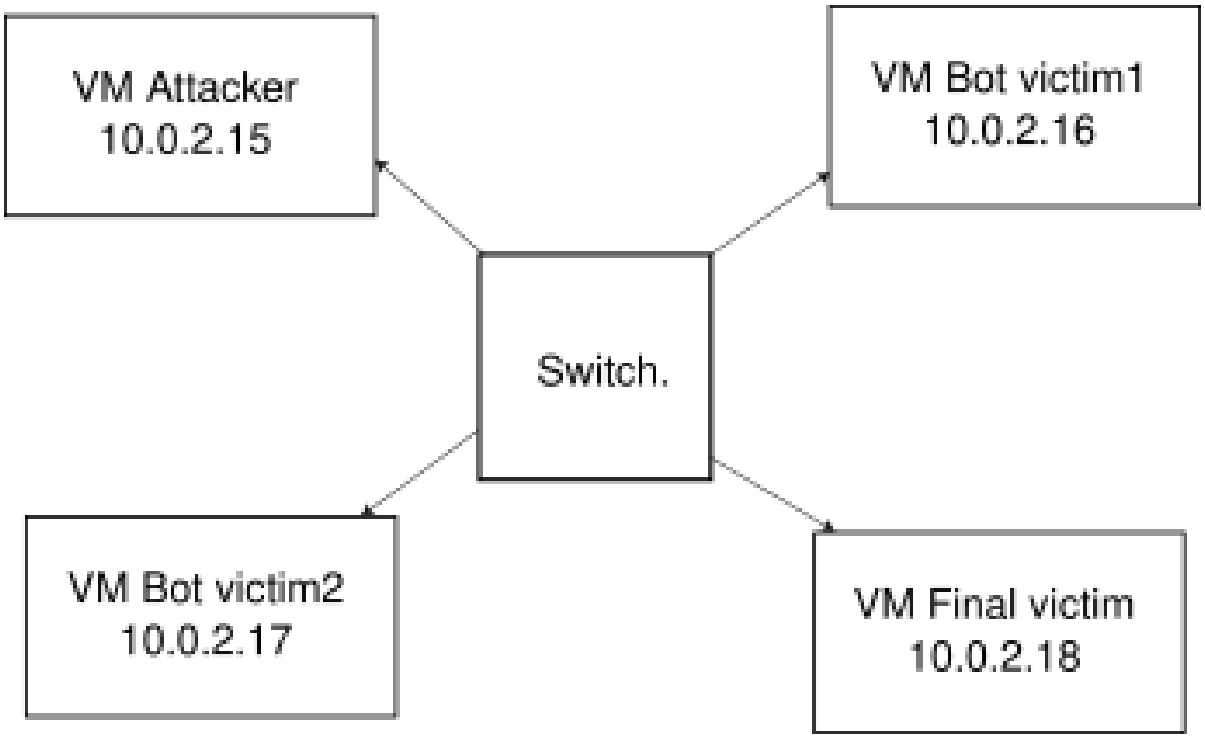
The goal of the client is to connect to the server, get its orders, and execute whatever the orders entail.

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 have 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 environmetn we first made a VM, with the dinit init system on it, aswell 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 vm and 1 more VM named vulnerableDOS. We called the first VM attacker, as it was going to be the initial aggresor 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 vicitm 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 ontrols 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/'.

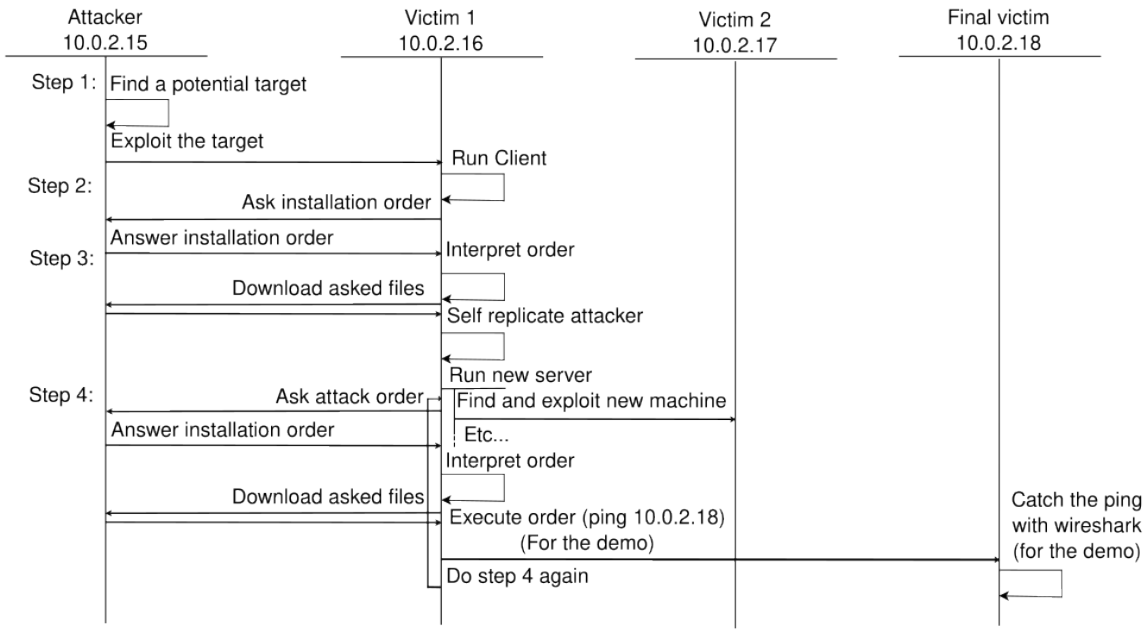
Step two. The client requests the server for its first instruction, called order1. The server answers with the file and the client saves it in '/tmp/botnet/conf/'.

Step three. The client reads the received order and every time it reachs a line begining with 'download', it requests the filesharing server for the respective filet to download which is saved to '/tmp/botnet/downloaded/'. Everytime the client reaches a line begining 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 is 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 ommission 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 conncetions 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 mutliple 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 lauch somes attacks. Althought it's usually used to lauch denial of services attacks, it can also be used to crack password or even mine cryptocurrency.
- **Other example:** Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque viverra pulvinar dui ut venenatis.

Refenrences

[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