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. 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 CVE-2014-6271 on hosts running an outdated version of Apache. Communication is encrypted with TLS thanks to OpenSSL and Rustls. By design, only 1 to 1 parent-child node relations are possible, owing to the targeting system implemented.

The exploit itself was just as important as the botnet itself. Without this, the hosts could never have been infected by its parent server, andso would never have joined the network of zombie machines.

Despite the age of the exploit, it is very likely that many machines and running this outdated version of Apache, meaning that our malware could potentialy ensnare multiple hosts, adding them to our botnet.

Keywords: Botnet, malware, virus, rust, TODO.

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.

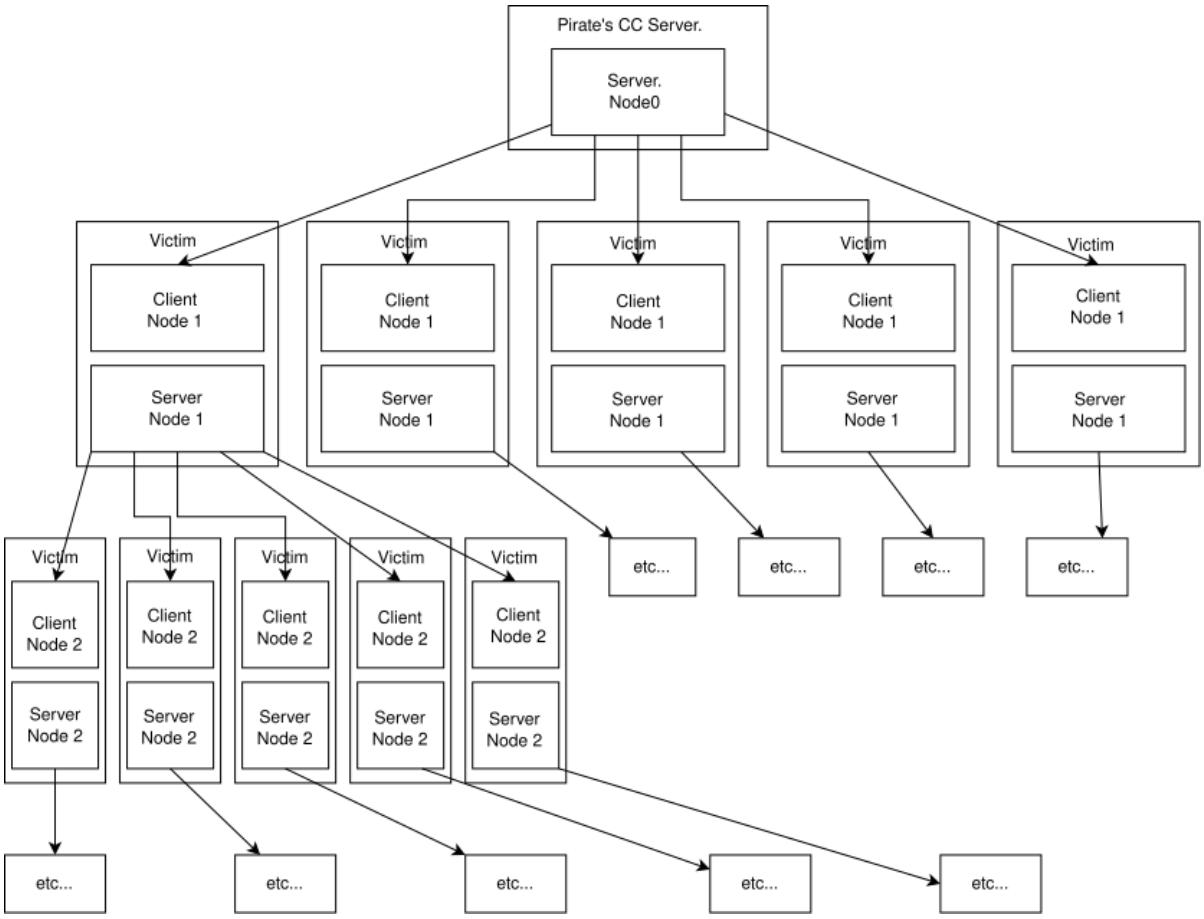
The first worm that we find was wanacry. (https://en.wikipedia.org/wiki/WannaCry_ransomware _attack) It's a bot which appeared on 2017 and targeted only Microsoft windows softwares. It was spread using a critical vulnerability of the SMB server that is on every windows machines. The way it's works is 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.

- Make a sandboxing environment, to test it, while been sure that it can't escape. We used qemu and unshare to make an environnment that couldn't reach the external network. In the end, it wasn't really usefull to make it, because we decided that for the demo, the victims ip should be knowed at launched time, so that we won't see unexpected things in the final demo.

- Find an exploit to use. It had to be a remote code execution, easy to setup, and easy to exploit, because it's not in the purpose of the project to find a new vulnerability, make a exploit or another crazy thing like that. We spend hours looking for the perfect exploit, and in the end, we find somebody else project on github, that was doing exactly what we were looking for. (https://github.com/opsxcq/ exploit-CVE-2014-6271) Thanks to opsxcq works, we had an easy vulnerability to exploit. We choosed shellshock, CVE-2014-6271. We find in opsxcq

repository, a vulnerable docker image to shellshock CVE-2014-6271, and an easy command to exploit it. The other two interested vulnerability that we found in his repos, but that in the end we choosed to not use, was CVE-2016-10033 and CVE-2017-7494. There were both unadapted to our needs because of how difficults they would makes us write the exploit. We also find Log4J, but in the end decided that we would better to abandon it, and try an easier vulnerability to exploit.

- Demonstrate the botnet. Like all botnet run things, we had to find a way to prove that the botnet has really taken control of several machine. For that, we made one more Vm, that we called vulnerableDOS. We simply run wireshark into, and once the botnet has taken control of the bots, it will make them ping our machine. At first, we wanted to make our botnet run an actual DDOS attack on the vulnerableDOS machine, but because it's not in the purpose of the project to make it run a complex attack, our goal is just to prove that we have indeed taken control of the vulnerables machines, we choosed to do only ping.

Obviously, we also had to make the actual botnet.

II.2. design explanations about the botnet.

Design and make the botnet was the real challenge. This is what we choosed to do. The architecture of our botnet will be a Server, that's gonna do three things. first, run the exploit on the victim. Second, wait for a response from the victim, to run a Command and Control server (CC), to deliver it's order. The first order in the demo will always be to replicate the server on the victim, and the second one to ping the vulnerableDOS vm. Third, the server will run a file sharing server (fs), so that everytime the CC server ask the victim to download a file, the file can download it.

On the CC and the fs server, we implement a TLS connection, using rustls. The certificates used are self signed certificates, and everytime the victim self-replicate the server, it will use the command openssl to generate the certificates. We were really lucky that on the vulnerable docker image that we find, there was openssl 1.0f installed. Maybe if the dockerimage was made one or two years earlier, there wouldn't be openssl installed, and we would had to abandon the idea to use encrypted connections. We were really lucky.

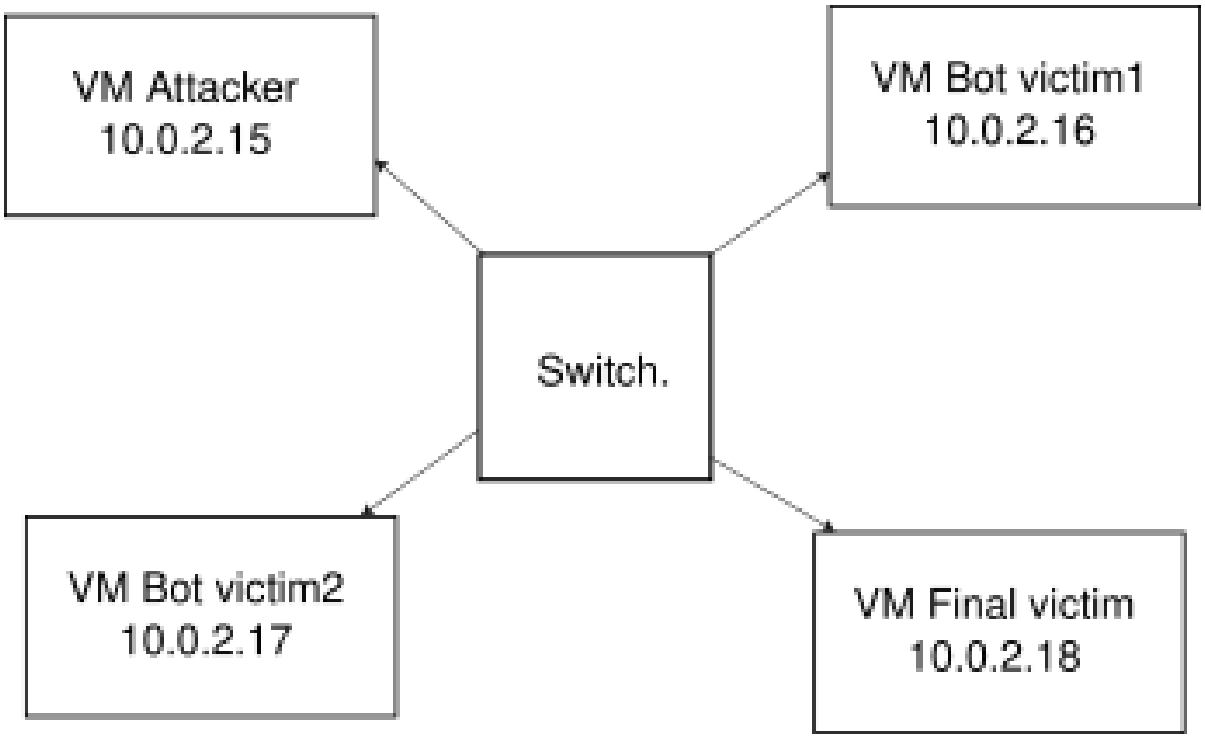
The architechture of our botnet will also be composed to a client. It's goal will be simply to connect to the server, get it's order, and execute it whatever it is.

The server will read in a conf folder it's TLS certificates, and the orders file that it will transmit to the victim. It will also share some scripts in a www folder, like 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 order to permit the client to once it has downloaded all the files it needed, to install them correctly, and self replicate the server.

II.3. lab setup.

To run the demo, we first made a base-vm, with runit-dinit on it, and only basic packages. Our goal was to know about everything installed, so that we don't loose time looking for where come a bug from, when it's for a default package. Then, we made 1 vm to be the head of our botnet, 2 vulnerable vm, so that when the botnet attack them, they can became bot, and a last one, named

vulnerableDOS, with wireshark installed on it.



Our choice to test the botnet was to make all bots to ping the vulnerableDOS vm. That means that our botnet will work if we manage to make bot 1 and 2 to ping vulnerableDOS vm.

II.4. Botnet communication.

This is the normal functionnement of the botnet. Step one. The server find an potential target. then, it runs a BASH script, called exploit.sh, which exploit the target. That means, exploit the vulnerability, in order to make the victim download and execute the client. In our case, the client is installed and ran in '/tmp/botnet/'.

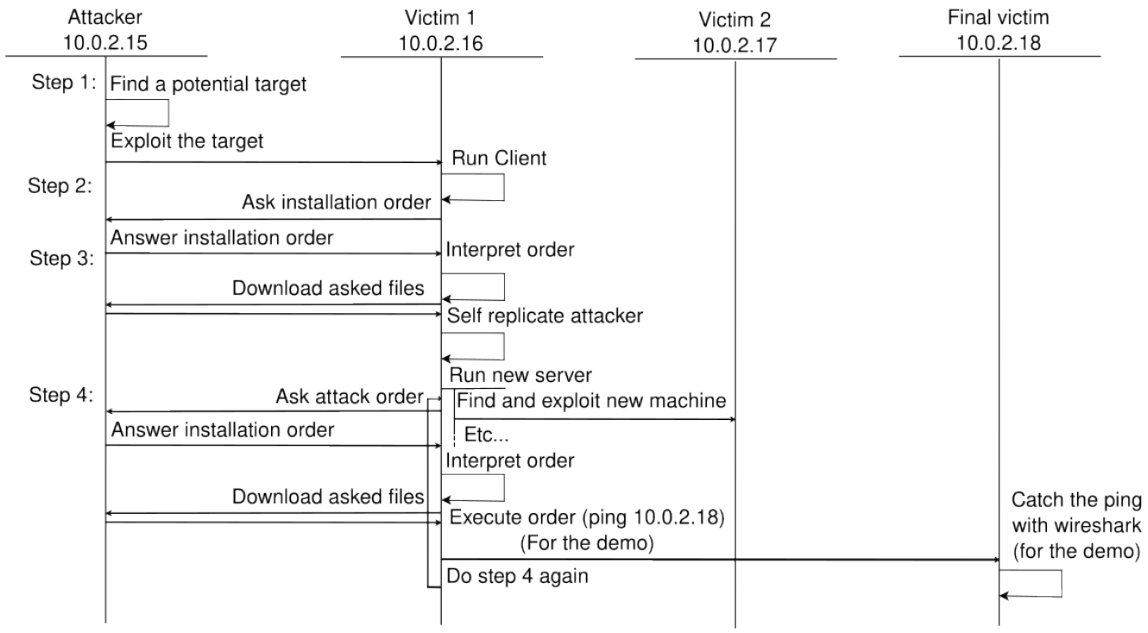
Step two. The client ask the server it's installed order, called order1. The server answer, and the client save it in '/tmp/botnet/conf/'.

Step three. The client read the received order, and every time it reach a line begining by 'download', it ask the filesharing server to give it, and put it in '/tmp/botnet/downloaded/'. Everytime the client reach a line begining by execute, it's execute the file.

Normally, the only executed file in the installation phase, it's the script 'installation.sh'. This script will copy all the downloaded files from '/tmp/botnet/downloaded' to their rightheous folders. It will then execute the server. The new server will look for other victims, and run again it's CC server and its fs server.

Step four. Finally, once the client had run another server, it will ask for it's 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‘ to prove that it has made a bot.



What the flowchart doesn’t says, is that every time it receive an order, it will save it, and make it accessible to the server so that is can relaie to the next node the orders. Also, everytime a file is downloaded, it will be made accessible, so that the next node will be able to download it.

Now that we have stated our actual reflection that we made before making the botnet, let’s look at what was the codding part of the project.

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

In the botnet creation phase, there are only two actors: the attacking server and the victim client. The attacking server, which is the root infector - attacker.cow, infects the first victim - vulnerable1.cow. Firstly it checks its targets, takes the first line of the file of targets, creates an ip address out of this and launches the exploit if the ip address is valid. In this case, the ip address belongs to vulnerable1.cow. Then, once the victim is successfully infected, it runs the client binary which is downloaded through the installation script<?> and connects to its parent node, which is the server running on attacker.cow. From this server, data is downloaded which will allow the vulnerable VM to start targeting other targets on the network which have a valid IP address relative to the list.

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. 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 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 where it does <thing>. Then the flow function is called where the connection is established with the file sharing server, receiving files form the file sharing server in a safe way.

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.

Rustls is set up and the ip address of the server is recorded and added to a file. 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. 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. At this point, the victim should be successfully hacked and be attempting a connection to the filesharing and C2 server, a connection which gets accepted. 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