Distributed Port Scanning

Zhecho D. Mitev
Department of Data Science and Knowledge Engineering
Maastricht University
Maastricht, The Netherlands
z.mitev@student.maastrichtuniversity.nl

Shir-Lee Kimelman
Department of Data Science and Knowledge Engineering
Maastricht University
Maastricht, The Netherlands
s.kimelman@student.maastrichtuniversity.nl

Julián R. Marrades Furquet
Department of Data Science and Knowledge Engineering
Maastricht University
Maastricht, The Netherlands
j.marradesfurquet@student.maastrichtuniversity.nl

***Abstract*—Scanning is a method used by attackers or defenders in order to find vulnerabilities for a potential attack. In this project we build a system with an intuitive a web user interface that allow a user to scan both IP addresses and ports. The IP scan can be done for one address or a range of them. The port scan is possible for a range of ports, with several scanning types: regular scan, SYN scan, and FIN scan. The system allows the scan to be distributed over and randomized over multiple scanning nodes in order to reduce detectability and increase efficiency. Several experiments were conducted with the system. The results show that higher open rate decreases execution time for IP scan, but increase it for port scan. Moreover, FYN scan has some timeout threshold, where below it there is a trade-off between speed and accuracy.**

***Index Terms*—network security, IP scan, port scan, distributed scanning**

I. Introduction

A cyber attack is a type of offensive maneuver that targets computer infrastructures and aims to access data without the authorization of its owner. An important part of the attacking process is the reconnaissance phase, in which the vulnerabilities of the system are explored. Although this phase does not cause any harm to the system it can potentially reveal important information, which may show the attacker how to take advantage of the weak points of the victim.

There exist different approaches for finding the vulnerabilities of a system and one of them is port scanning. It is used to discover hosts and services on a computer network by sending packets and analyzing the responses. Each host has a unique IP (Internet protocol) address, which is used as an identification of the host. It also has access to logical addresses called ports. By definition, a port scanner checks whether a subset of ports, belonging to a specific IP, are opened or not. This approach is often applied to determine available services on a remote machine, however it can be used by an intruder who is preparing for an attack, as well. Previous study indicates the growth of incoming scanners traffic, maximum number of scans per scanner, and the speed (scanning rate), during the period 1994-2006 [2].

For an attacker it is very important that his port scan is not detected by the victim. In the case when the scan is not performed in a stealthy way, it can alert the security infrastructure of the victim. Usually, simple scans can be easily tracked by a defending system and a further attack can be prevented. For example, scanning with a single source IP address or in a sequential order can cause suspicious network traffic. Former research shows that a distributed approach can be vital for escaping from a detection of scanning activities [4]. In this strategy, attackers use multiple IP addresses to send its port probes. This allows the intruder to investigate a large part of the victim's system without causing massive traffic from one IP. In order to produce a stealthy and effective tool, we create a distributed scanner, which assigns tasks among numerous hosts, each scanning a subset of targets. All hosts receive orders independently from each other and the targets are scanned in a random order, so that the process cannot be easily observed by a defender. The first option that is available in our tool is IP scanning, which can be performed both on a single or on a range of IPs. We show a vertical port scanning, which has several modes (TCP full, TCP SYN and TCP FIN) and those are compared in the experiment section of this report.

II. Background

A. IP Scanning

An Interent Protocol (IP) address represents a device, that belong to a network and uses the Internet Protocol for communication under the internet layer.

An IP scanner will try connect with an IP using the Address Resolution Protocol (ARP). If there is a reply, then it is alive. Otherwise, we assume it is not.

B. Port Scanning

A port is a communication endpoint, used in the transport layer. There are 65,536 TCP and 65,536 UDP ports on a machine, which are split into three ranges by the Internet Assigned Numbers Authority (IANA) [1] :

1) Well Known Ports: from 0 to 1023
2) Registered Ports: from 1024 to 49151

3) Dynamic and Private Ports: from 49152 to 65535.

A port scan checks a network host for open ports that can be a victim for future attack. It does not attack the system itself, but gather information for the potential attack. It is mostly for TCP ports, since they give more relevant information than UDP ones. There are multiple scanning categories [3]: stealth scan, SOCKS port probe, bounce scan, TCP scan, UDP scan. More specific types includes:

- Regular scan (TCP connect()): A scanner sends a packet with a SYN flag to establish a connection with a port. If it receives a packet with a SYN and ACK flags, then the port is open. Otherwise, we assume the port is closed. If the port is open, a packet with an ACK flag should be sent back to complete the handshake (and data can be delivered).
- SYN scan: Here the scanner sends a packet with SYN flag to a port. If it receives a packet with a SYN and ACK flags, then the port is open. However, if the packet has a RST flag, then the port is closed. If the port is open, the scanner will send a packet with a RST flag.
- FIN scan: In this case, the scanner send a packet with a FIN flag. If a packet with a RST flag is received, then the port is closed. Otherwise, the port is open.

III. Methodology

A. Overall architecture

The project included a server with web User Interface (UI), that allows distributed IP scanning and port scanning. The scanning is distributed over multiple scanning nodes.

The nodes register themselves to the server, by establishing a TCP connection (three-way handshake). When the server has a scanning task, it distributes it randomly over the registered nodes that are available and listening.

The sub-tasks are transferred via packets, with data representing the instructions (i.e. which scanning to perform and what should be scanned).

After a sub-task is given to a node, it starts the scanning it was ordered to (described in the background section). When the scanning is done, the node sends the server the results (alive or not for each of the addresses/ports. The results are sent via packet(s) with the data. The server communicates the aggregated results back to the user via the UI.

B. User Interface

A UI was created for easy and intuitive communication of the server with the user, a potential attacker or defender who wish to apply a scanning.

The UI has two modes:

- Listening: In this mode the server is listening to available scanning nodes that register themselves. The UI will display all the nodes currently registered to the server (for instance, see Figure 2).
- Apply task: In this mode the server stop listening to new registrations in order to apply a task it gets from the user.

These modes can be applied by the 'listening' button.

For the scanning task the UI allows the user to (1) choose the type of scanning (2) define what should be scanned. Then the output of the scan is displayed on the screen together with the time the scanning took (in milliseconds).

For instance, in Figure 3 the user chose to do an IP scan for a range of IP addresses. The user fill the wanted range of 172.17.0.5-172.17.0.15 and press the button 'which are alive?'. The scan is applied, and the UI displays for each of the IP addresses whether it is alive, and reporting that the scan took 11424 milliseconds to perform the scan.

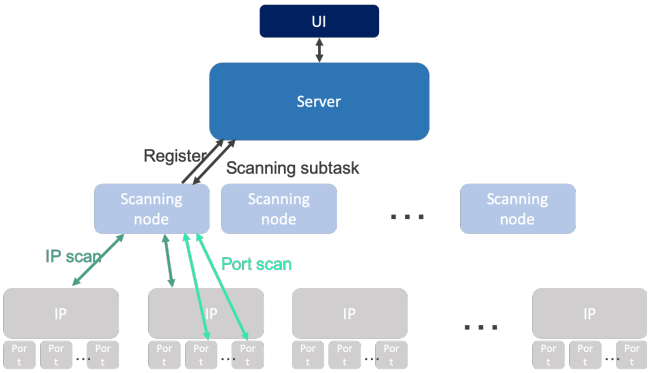

Fig. 1. Architecture

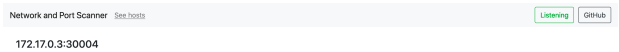

Fig. 2. UI showing the registered scanning nodes

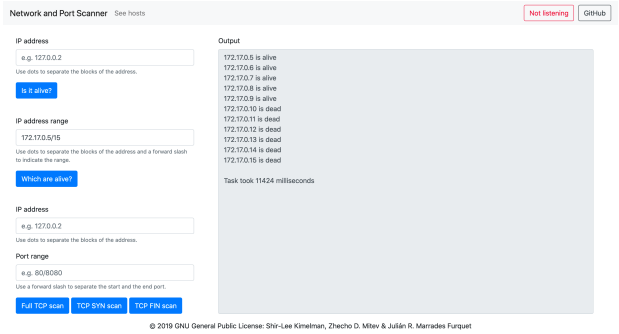

Fig. 3. UI when applying a scanning task

C. Infrastructure

For the server we employed a Flask application which is dockerized into a Ubuntu 18.04 container. For the scanning

nodes a simple Ubuntu 18.04 docker container. Every docker container gets assigned a different IP address, but for docker port forwarding, we had to make every scanning node listen to a different port.

D. Scanning node registration

The UI allows the server to switch between two main states: listening for new host registrations, and not listening. When a host container is activated, it will try to perform a TCP-IP 3-way handshake with the server to register itself. In order for the server to know which port the host is listening to, it will use the 'seq' parameter in the TCP header of the initial SYN packet to indicate which port is open to incoming connections. After the handshake is completed, all communication between server and nodes will employ the TCP-IP protocol.

E. Instruction communication

When an instruction is issued from the UI, it is passed to the Python backed and distributed among nodes using the TCP-IP protocol.
In order to make the data transmission as efficient as possible, the instructions should be transformed into bytes and appended to the packet.
The data in the packet is represented as follows:

- Mode: The first byte is the mode of scanning. There are five modes available: (1) IP scan for an address, (2) IP scan for a range of addresses, (3) regular scan for range of ports, (4) SYN scan for range of ports, (5) FIN scan for range of ports.
- IP address: The next three bytes represent the first three parts of an IP address for the scanning.
  - One IP: In the case of one IP address (i.e. all modes except of mode 2), there will be one additional byte as the forth part of the IP address.
  - IP range: In the case of IP scan for a range of addresses (i.e. mode 2), any additional byte is the forth part.
- Ports range: The last bytes represent the ports (bytes 6 and onward). Each port is represented by two bytes. This is relevant for port scanning modes.

The reason of representing both IP addresses and ports as a full list (and not indicating only start and end) is that it makes it more convenient for implementation of the random distribution (see III-F).

F. Distribution of tasks

Except for converting the information to bytes, the server has the responsibility to decide how much packets should be sent to each host. There are several scenarios which are treated differently in our case:

1) The most simple case that can occur is when the ports to scan are less that the available clients. In this case, we start sending one packet for each port and some of the host don't even need to perform any work.
2) If the total ports are divisible by the number of scanning nodes, then we distribute the tasks equally among the nodes, so that the job is finish in shortest time. In addition, a shuffling function is included as an option. It determines a random set of ports that each host has to scan. In this way, the execution is more stealthy.
3) The last possibility is when the tasks cannot be equally distributed (e.g. 4 host, 6 ports). In such occasion, we compute the number of ports per host (which is 1.5 in this example). Then, the first node will scan only one port and will ask the second one to scan the rest (0.5 ports) for it. Thus the second node receives a request to scan 0.5 + 1.5 = 2 ports. In short, every fraction that is left is passed to the next host. This approach is a very efficient for achieving a comparatively equal distribution of the tasks and prevents a situation in which one node has much more workload than another one.

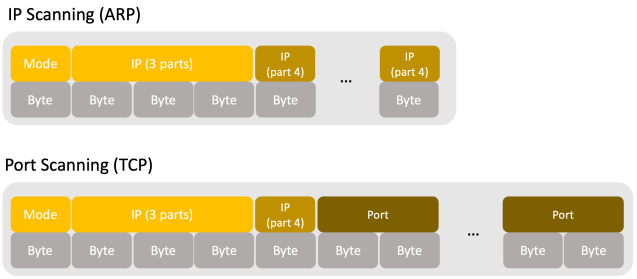

Fig. 4. A subtask packet data

G. Result communication

After a scanning node has fulfilled its instructions, it has to encode the response in bytes and send it to the main server. No matter the mode of scanning, the results can be stored in a string of binary numbers, where 0 means dead and 1 means alive. For instance, if the node is asked to scan 10 ports, he could have the following outcome: "0110100010", meaning the first port is dead, the second is alive, the third alive, and so on. We can encode this result making divisions of length 8 and transforming the bits into bytes (e.g. "00000011" -¿ 2 -¿ 0x02). Since the length of the result may not be a multiple of 8, we shall append enough "0"s to the end of the bitstring to satisfy this condition.
Once the result is encoded in bytes, it is sent back to the server via TCP-IP.

H. Result decoding by server

Once the server receives the response of a node, it transforms the bytearray into a bitstring (appending byte by byte). Let n be the number of ports/IPs which the server told the node to scan. The server grabs the first n bits from the bitstring and discards the rest (used by the node to fill bytes). Since the server knows the instruction which it gave to the node, it can retrieve which ports/IPs are dead or alive using the bitstring.

IV. Experiments

We designed experiments to examine the scanner in the following perspectives:

1) The impact of the open rate on performance: We define open rate as the ratio of alive instances and total instances. Therefore, the possible values for open rate are in range (0, 1).
   For the IP scanning, our hypothesis states that the time to execute a task is reduced when the open rate is increasing. The reason behind it is that if an IP address is open then it receives a response, and if it is closed there is a need to wait for a timeout. Therefore, the higher the open rate, the less closed ports, and hence less ports to wait for a timeout.
   For the port scanning, we hypothesized that the higher the open rate the more time it take to scan. This is since if a port is open than we need to act (send back a packet) or even wait for a timeout (in the case of a FIN scan). Hence open ports will take more time.
2) The impact of reducing the timeout on performance: As mentioned, in FIN scanning a port is open if there is no answer (i.e. timeout). Therefore there is a trade-off between the timeout and accuracy. We hypothesized that under some threshold we will lose accuracy, such that the lower the timeout the lower the accuracy performance.

Each of the experiments are evaluated on all scanning types (i.e. modes). We perform IP scanning on a range of IPs and port scanning of a range of ports of one IP, as regular, SYN, and FIN modes are all tested. Each setting was tested 3 times and then averaged, in order to reduce noise and gain a better evaluation.
In all experiments the following default values (unless stated otherwise) were used: number of scanning nodes of 5, open rate of 0.3, timeout for checking whether is alive of 1 second, timeout of instructions to the scanning node of 30 seconds.
Performance is evaluated by two measures: Firstly, the time it takes to complete the task. Secondly, the accuracy rate, i.e. how many of the open and closed ports were classified correctly.

V. Results

A. Experiment 1: impact of open rate

The achieved accuracy is 100% in all cases.

In the IP scanning, the higher the open rate, the less time it takes for scanning.

In all port scanning, the higher the open rate, the less time it takes for scanning. In the case of FIN scanning this relation the slope is even higher (steeper trend line).

B. Experiment 2: impact of reducing the timeout in F

For a timeout of 0.1 and above, the accuracy of 100% is always achieved. However, From timeout of 0.01 and below we can see that reducing the timeout also reduces the accuracy.

VI. Discussion

Figure 5 shows us the relationship between the fraction of alive IP addresses and the average time it takes to perform an ARP scan. We observe that a linear relation fits well the obtained results, so a best fit line is added, resulting in the equation $y = -10.164 + 12.199$, where y is the time in seconds and x is the rate of alive IPs. The large value of the R^2 coefficient indicates the high likelihood of a correlation. Knowing that detecting a closed IP takes longer than detecting an alive open because in the former case it is needed to wait for the timeout, an inverse relation between x and y was expected. We can confirm the hypothesis looking at the negative slope of the best fit line.

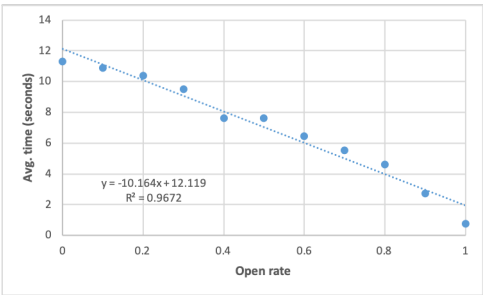

Fig. 5. Performance for IP scanning with different open rates

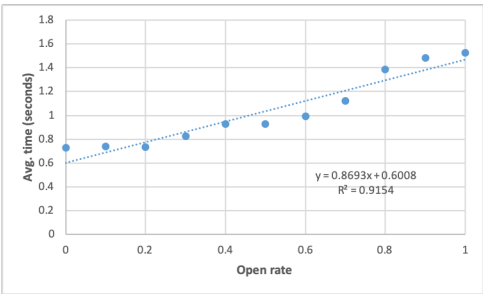

Fig. 6. Performance for regular port scanning with different open rates

Timeout	Accuracy
1	100
0.1	100
0.01	95
0.005	84
0.002	80
0.0015	72
0.0012	75
0.001	52
0.0001	46
0.00015	36
0.00001	15
0.000001	0

TABLE I
PERFORMANCE FOR FIN PORT SCANNING WITH DIFFERENT TIMEOUT

Figure 6 let's us determine the time which a TCP connect scan will take given the rate of open ports. A high value of R^2 motivates the use of a best fit line described by $y = 0.8693x + 0.6008$. The positive slope confirms our hypothesis and its reduced value indicates the short computational time which is required when a port is open to complete the 3-way handshake.

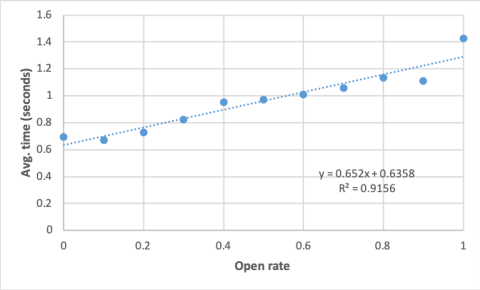

Fig. 7. Performance for SYN port scanning with different open rates

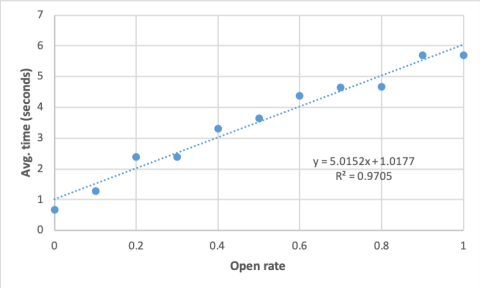

Fig. 8. Performance for FIN port scanning with different open rates

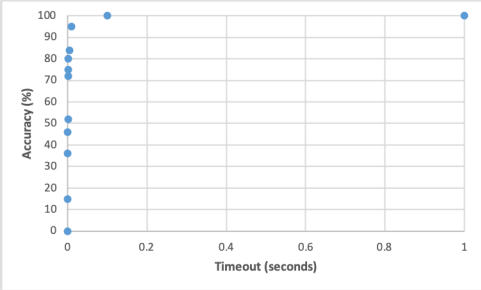

Fig. 9. Performance for FIN scanning with different timeout values

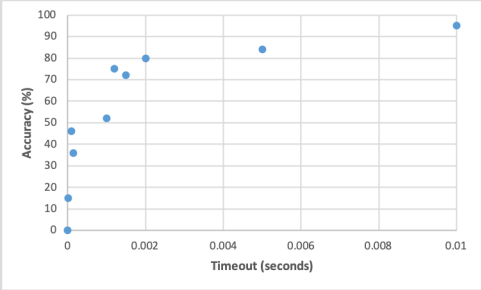

Fig. 10. Performance for FIN scanning with different timeout values (with zoom in)

Regarding TCP SYN scans, Figure 7 shows a value for R^2 which is very similar to the one showed for TCP complete in Figure 6. Moreover, the positive slope in $y = 0.652x + 0.6358$ proves our expectations.
When comparing the TCP SYN and TCP complete equations, we find that the slope for TCP SYN is significantly smaller than the TCP complete one. This is due to the fact that a TCP SYN scan performs only half of the handshake, while a complete scan performs it entirely. Looking at the equation constants, the difference between them is 0.035 seconds. We would expect such a small difference because in both TCP SYN and TCP complete scans, when a port is closed and an RST flag is received, no further communication with the victim is established.

For TCP FIN scans, the plot in Figure 8 shows a higher R^2 than in TCP SYN or complete scans, indicating a more reliable relational model by the equation $y = 5.0152x + 1.0177$. Once again, a positive slope confirms our hypothesis of a positive proportional relationship between open port rate and average scan time. What is more, it was expected for slope to be significantly larger than for TCP SYN and complete scans, due to the fact that for every open port, the node will spend at least 1 second in the scan (waiting for timeout).

Finally, we tried to make SYN FIN scans faster by means of reducing the timeout which the nodes employ for communication with victims. Table V-B shows that a timeout $t \leq 0.1\ s$ is needed to fall below 100% accuracy in the scan. By $t = 0.001\ s$ the accuracy has dropped to 52% and $t = 0.000001\ s$ makes the scanning nodes identify all closed ports as open (0% accuracy).

VII. Limitations

One of the main limitations of our work is that even though, we are applying a distributed approach, it would be very simple for a defender system to trace our scan. An untraceable scanner would require a much more complex implementation, which also can be only performed by a network expert.

VIII. Related Work

A. Scanning techniques

Various port scanning techniques can be categorized into five main categories [3]:

- Stealth scan: Trying to be undetected by auditing tools, by sending packets with stealth flags.
- SOCK Port Probe: Scanning SOCK ports since many users misconfigure them, and they may allow access to other Internet hosts.
- Bounce Scan: Trying to use the vulnerability of the FTP.
- TCP Scanning: Trying to find the ports that accept connection. In our implementation, only the full TCP completes the connection. The rest of the modes only send a request to check whether the target is alive. Such scanning is harder to detect, since it is not logged. This category includes SYN, and FIN scans, that were used in our approach. Other possible TCP scan methods are Reverse Identification, XMAS, TCP fragment, etc.
- UDP Scanning: Trying to find open ports related to the UDP. However, it is hardly used, since UDP is connectionless and easily blocked.

B. Distribution

The scanning techniques can be divided by their distribution [3]. A Single source scan has a relation of ”one to many”, where one machine scans several machines. It can be divided into four [11]: (1) vertical, to scan for port(s) in a single machine (IP address), (2) horizontal, to scan for a single port in multiple machines, (3) strobe, to scan for multiple ports in multiple machines, (4) block, to scan for all ports in multiple machines.
A distributed scanner has multiple machines that are scanning. It can have ”many to one” relation, where they scan one machine, or ”many to many” relation, where they scan for multiple IP addresses or ports. A distributed scan can be performed using several techniques, such as (1) rate-limited technique, which limits the number of packets sent for scanning, (2) random or non-linear, which randomizes the order of the IP-port combinations that need to scanned. Moreover, it can randomize the time delay for each probe packet [3].
Even though those distribution techniques improve the quality of a scanner in general, there is a significant research about the detection of such scanners. Robertson et al. [9] suggests approaches to detect distributed port scans based on the location of the source IPs. That paper gives the possibility of revealing strategies that do not spread their sources across multiple networks, far apart from each other. In general, there are many different approaches that try to cluster packets according to a similar feature or detect footprint patterns based on the goals of the intruder [5, 10]. Such techniques are theoretically very efficient against malicious probing.

C. Tools for network scanning

In network security there are tools that can explore systems and give information about the state of its instances. Those tools are mainly used by ethical hackers to test a defending system that prevent such scans or by administrators for maintenance purposes. Actual attackers rarely use those tools because they are comparatively simple and can be easily traceable by the existing scanning detector [5, 6].

1) Nmap: One of the most famous network scanners tools, that Gordon Lyon released it in 1997, is called Nmap [7]. It is used by network administrators to look for alive hosts or services by scanning a network. Even though the tool is very popular, it does not allow for distributed scanning. However, several extensions of the Nmap have been created and one of them is Remote Nmap (RNmap) [8]. It is a client server tool ,implemented in Python, that settles a distributed scanning environment. Both scanners come with many options for carrying out various security scans of a network.

2) Other tools: There are many other open-source scanners that are designed to be fast. An example is *Angry IP* [1] which can scan NetBIOS information (computer name, MAC addresss) and favorite IP address ranges, apart from the IP address and port scanning it can perform. Other very popular tools are Unicornscan, Netcat, Zenmap, etc.

IX. Conclusions

In this paper we have developed a distributed IP and port scanning architecture in which instruction commands come from the UI of a central server and are forwarded to several scanning nodes, which are the active scanners. The reason for such a distribution comes from the fact that if a given IP is detecting incoming packets for port 0, then port 1, and so on, from the same (attacker) address, it can easily detect a threat. Thus, we randomize the IP/port orders and give subsets of them to many scanning hosts, which perform the task from different IPs, applying a stealth layer to the scan.

Firstly, we introduce the reader to the basics and IP and port scanning, making it easy to follow the rest of the paper. Secondly, we describe our architecture: the user interface, server, scanning nodes, TCP-IP server-node communication, task distribution, etc.
Thirdly, we presented the experiments which were performed in the platform. The results and discussion of the experimental phase follow in the paper.
It was observed that, as expected, IP scanning time decreased as the rate of alive victims increased, due to the fast reply and non-necessity to wait for a response timeout. Moreover, TCP complete, TCP SYN and TCP FIN scans follow the predicted behavior, i.e. increase in time with increase in port open rate. It could be observed that the time growth slope in TCP complete is steeper than in TCP SYN, due to the completion of the handshake. For TCP FIN we observe a high value for the time growth slope. This is because when a port is open, the node has to wait for the timeout to declare that it’s state.
After, we tried to speed up the TCP FIN scan by reducing the timeout value. It was noticeable that when the timeout values started to get closer and closer to 0, the accuracy decrease rate starts to speed up, until 0% accuracy is finally reached.

Finally, the limitations of our work were described and other related work was explored. Scanning techniques such as Stealth scan, SOCK Port Probe, Bounce Scan and UDP scanning were explained, existing distributed scanning and distributed scanning reconnaissance techniques were tackled and other tools for network scanning such as *nmap* and *Angry IP* were briefly discussed.

We believe that this paper presents an approach which tries to tackle existing difficulties in distributed port scanning, while leaving enough space for other scholars to extend our architecture and build new approaches over the solid base we have managed to create.

REFERENCES

[1] Angry ip scanner. https://angryip.org/about/. Accessed: 2019-10-14.

[2] Mark Allman, Vern Paxson, and Jeff Terrell. A brief history of scanning. In *ACM Internet Measurement Conference (IMC07*, 2007.

[3] Monowar H. Bhuyan, D.K. Bhattacharyya, and J.K. Kalita. Surveying port scans and their detection methodologies. *Comput. J.*, 54(10):1565–1581, October 2011.

[4] Evan Cooke, Michael Bailey, Z. Morley Mao, Danny Mcpherson, David Watson, and Farnam Jahanian. Toward understanding distributed blackhole placement. In *In Proceedings of the 2004 ACM Workshop on Rapid Malcode (WORM-04*, pages 54–64. ACM Press, 2004.

[5] Carrie Gates. Coordinated scan detection. In *In 16th Annual Network and Distributed System Security Symposium*.

[6] Urupoj Kanlayasiri, Surasak Sanguanpong, and Wipa Jaratmanachot. A rule-based approach for port scanning detection. 11 2002.

[7] Gordon Fyodor Lyon. *Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning*. Insecure, USA, 2009.

[8] Angela Orebaugh and Becky Pinkard. *Nmap in the enterprise: your guide to network scanning*. Elsevier, 2011.

[9] Seth Robertson, Eric V. Siegel, Matthew Miller, and Salvatore J. Stolfo. Surveillance detection in high bandwidth environments. *Proceedings DARPA Information Survivability Conference and Exposition*, 1:130–138 vol.1, 2003.

[10] Himanshu Singh. Distributed port scanning detection. *Master's Projects*, 01 2009.

[11] Stuart Staniford, James A. Hoagland, and Joseph M. McAlerney. Practical automated detection of stealthy portscans. *J. Comput. Secur.*, 10(1-2):105–136, July 2002.