San Jose State University
SJSU ScholarWorks

Master's Projects | Master's Theses and Graduate Research

1-1-2009

Distributed Port Scanning Detection

Himanshu Singh
San Jose State University

Follow this and additional works at: http://scholarworks.sjsu.edu/etd_projects

Recommended Citation
Singh, Himanshu, "Distributed Port Scanning Detection" (2009). *Master's Projects.* Paper 142.

This Master's Project is brought to you for free and open access by the Master's Theses and Graduate Research at SJSU ScholarWorks. It has been accepted for inclusion in Master's Projects by an authorized administrator of SJSU ScholarWorks. For more information, please contact scholarworks@sjsu.edu.

DISTRIBUTED PORT SCANNING DETECTION

A Writing Project

Presented to

The Faculty of the Department of Computer Science

San José State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Himanshu Singh

May 2009

© 2009

Himanshu Singh

ALL RIGHTS RESERVED

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

Dr. Robert Chun

Dr. Mark Stamp

Dr. Agustin Araya

APPROVED FOR THE UNIVERSITY

ABSTRACT

DISTRIBUTED PORT SCANNING DETECTION

by Himanshu Singh

Conventional Network Intrusion Detection System (NIDS) have heavyweight processing and memory requirements as they maintain per flow state using data structures like linked lists or trees. This is required for some specialized jobs such as Stateful Packet Inspection (SPI) where the network communications between entities are recreated in its entirety to inspect application level data. The downside to this approach is that the NIDS must be in a position to view all inbound and outbound traffic of the protected network. The NIDS can be overwhelmed by a DDoS attack since most of these try and exhaust the available state of network entities. For some applications like port scan detection, we do not require to reconstruct the complete network traffic. We propose to integrate a detector into all routers so that a more distributed detection approach can be achieved. Since routers are devices with limited memory and processing capabilities, conventional NIDS approaches do not work while integrating a detector in them. We describe a method to detect port scans using aggregation. A data structure called a Partial Completion Filter(PCF) or a counting Bloom filter is used to reduce the per flow state.

DEDICATION

To my family. Thanks for all the support!

ACKNOWLEDGEMENTS

I would like to thank Dr Chun for the guidance and support that he offered my through the process of the writing project. I would like to thank Dr Stamp for encouraging me to develop a distributed scanner as a class project. This work evolved from wearing both black and white hats. I am grateful to Dr Araya for his valuable suggestions.

CONTENTS

CHAPTER

1 INTRODUCTION 1

1.1 Scalable port scan detection 2

1.2 Overview of the report 3

2 BACKGROUND 4

2.1 Port scanning 4

2.2 Classification of scans 6

3 MOTIVATION 7

3.1 Design considerations 7

3.2 Related work 8

4 APPROACH 11

4.1 Simulation environment 11

4.2 TCP Scanner 12

4.3 Packet sniffer 15

4.4 Detector 18

4.4.1 Patterns in TCP packet traffic 20

4.4.2 Partial Completion Filter (PCF) 21

4.5 Network Topology 21

5 RESULTS 24

6 CONCLUSION 28

BIBLIOGRAPHY 31

LIST OF TABLES

Table

4.1 Fields extracted by the packet sniffer 16

5.1 Results of two scanners and two targets 25

5.2 Results of four scanners and two targets 27

LIST OF FIGURES

Figure

2.1 Conceptual geometric pattern of common scan footprints [9] 6

4.1 Simple and compound modules 11

4.2 TCP State diagram [20] 13

4.3 IP v4 header [2] 17

4.4 TCP header [2] 18

4.5 Prototype IDS within router r3 19

4.6 Multiple stage Partial Completion Filter [13] 22

4.7 Port scan experimental setup in OMNeT++ with 2 scanners, 2 targets, and no background traffic. 23

5.1 Experiment setup with two scanners and two targets 24

5.2 Experiment setup with four scanners and two targets 26

Chapter 1

INTRODUCTION

Scanning activity is regarded to be a threat by the security community - an indicator of an imminent attack. Panjwani et al found that 50% of all scanning activity was followed by an attack[16].

Incidents of computer break in and sensitive information being compromised are fairly common. There are substantial financial gains to be made from electronic theft of data . Utility providers using information technology for efficient management of resources across increasingly greater regions are vulnerable to service disruption by electronic sabotage of their centralized systems[15].

> Attack programs search for openings in a network, much as a thief tests locks on doors. Once inside, these programs and their human controllers can acquire the same access and powers as a systems administrator[10].

Government computers were the target of an espionage network which compromised thousands of official systems worldwide [7]. The attacker with the greatest technical sophistication is the professional criminal or the cyber terrorist. A sophisticated adversary is risk averse and may go to great lengths to hide their tracks[1]. This is because detection may provoke a response by the defender – either retaliation or up gradation of the defenses. One of the tactics used in warfare is reconnaissance

or information gathering. Reconnaissance can be non technical - social engineering, dumpster diving or technical – scanning the target's network, monitoring traffic[21].

The method of determining services available on a computer by sending packets to several ports is called port scanning[8]. Further communication on the ports that services are available can determine the vulnerability to any available exploit and is termed vulnerability scanning. The scanning packets traverse the target network and so are visible to any network application such as an Intrusion Detection System (IDS). This may cause them to be detected. Avoiding detection by IDS can be as simple as insertion of a time delay between scanning packets, thereby defeating most thresholding based IDS algorithms. However this is not efficient as it slows down the scanning activity. For a more efficient approach, other methods have evolved like coordinated / distributed port scans. These divide the target space among multiple Source IPs (SIPs) such that each SIP scans a portion of the target. The IDS may not detect this activity due to the small number of connection attempts, or if it does, then it may not be able to detect the collaboration between the source machines.

Early detection and reaction to potential intruders is made possible by the detection of port scans, stealthy or co-ordinated port scans. Cohen [5] determines optimal defender strategies by simulating computer attacks and defences. He finds that responding quickly to an attack is the best strategy that a defender can employ. A quick response is better than having a highly skilled and multilevel defence in place, but an increased response time to an attack.

1.1 Scalable port scan detection

In a nutshell, we would like to use aggregation techniques to scalably detect distributed port scanning activity by fast spreading Internet worms and validate the

detector using a simulator[24]

1.2 Overview of the report

Chapter 2 is a primer on types of scans and detectors. In Chapter 3 we present our motivation and related work in port scan detection. Chapter 4 introduces the detector that we have built. Chapter 5 is an analysis of the data generated by the simulation of the detection algorithm. Our conclusion is presented in Chapter 6.

Chapter 2

BACKGROUND

Port scanning is a method of determining the available services on a computer by sending packets. It is generally viewed as a reconnaissance activity or information gathering phase distinct from the attack phase. This implies that there will be a gap between the scan and the attack. But there are no technical reasons for separating the reconnaissance activity with the attack phase when fast propagation is a key consideration. This can be achieved with an integrated scan and exploit tool. There is a trade off between between speed and stealth of the scanning activity. The motivation of the attacker dictates the choice between speed and stealth. Fast propagation is a kind of brute force scan/attack and is easily detected by the target network security personnel. Some scanning activity is immediately followed by an attack. This is probably to take advantage of zero day exploits.

2.1 Port scanning

A listening service on a network host is referenced by the combination of its host IP address and the bound port number. A port is a logical address on a machine. There are 65,536 TCP and 65,536 UDP ports on a machine. These are split into three ranges by the Internet Assigned Numbers Authority (IANA)[18] :

(1) *Well Known* Ports, from 0 through 1023

(2) *Registered* Ports, from 1024 through 49151

(3) *Dynamic* and/or *Private* Ports, from 49152 through 65535

> "Port Scanning is the process of identifying some or all open ports (listening services) on one or more hosts.[14]"

A port scan may be the precursor to an actual attack, so it is essential for the network administrator to be able to detect it when it occurs.

A simple port scan by itself does not harm the host as it concentrates on the *Well Known Ports*, and is done in a *sequential* manner. If, on the other hand, enough such *simultaneous* connect attempts are made, the host's resources may get exhausted and its performance adversely affected, as the connection state has to be maintained. Clearly, this can be used as a Denial Of Service (DOS) attack.

In order to detect and prevent port scanning, various Intrusion Detection/Prevention System (IDS/IPS) are used. IDS/IPS identifies multiple connection requests on different ports from a single host and automatically blocks the corresponding IP address. The best example of this kind of IDS/IPS is Snort [14]. Distributed port scanning is used to evade detection and avoid the corresponding black listing of the source machine by the target host/network.

A conventional port scan targets a single or a few chosen hosts,with a limited subset of carefully chosen ports. This type of scan is slow and and generally used on pre chosen targets, so its IP coverage focus is narrow. A specific type of port scan called a *sweep* targets whole IP ranges, but only one or two ports. Here the objective is to quickly cover as many hosts as possible, so its IP coverage focus is broad. This sweep behavior is generally exhibited by a worm or an attacker looking for a specific vulnerable service.

2.2 Classification of scans

Scans can be classified by their *footprint* which is nothing but the set of IP/port combinations that is the focus of the attacker. The footprint is independent of how the scan was conducted or the *script* of the scan [22]. Staniford et al note that the most common footprint is a *horizontal scan*. They infer that this is due to the attacker being in possession of an exploit and interested in any hosts which expose that service. This footprint results in a scan which covers the port of interest across all IP addresses within a range. Horizontal scans may also be indicative of a network mapping attempt to find available hosts in a range of IP addresses. Scans on some or all ports of a single host are termed *vertical scans*. The target is more specific here and the purpose is to find out if the host exposes any service with an existing exploit. A combination of horizontal and vertical scans is termed a block scan of multiple services on multiple hosts [22].

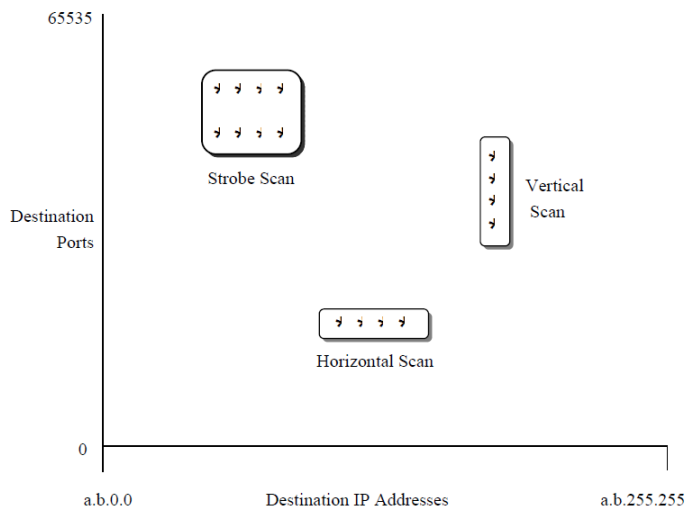

Figure 2.1: Conceptual geometric pattern of common scan footprints [9]

Chapter 3

MOTIVATION

We developed a distributed port scanner which used proxy response fingerprinting based on a presentation at the RSA 2006 conference [14]. We used the free open application proxy Squid [6] as the intermediary and implemented the scanner in Perl.

3.1 Design considerations

There are a lot of variables that require careful consideration while designing a detector. We make the following assumptions about the operating conditions of the detector

- A medium to large size network with multiple gateways and quite possibly delegated administrative authority.
- The core network administrators require fast detection and logging of any distributed scanning activity. However, there will be no automated response to any flagged scanning activity. (No auto ban or blacklisting) The flagged activity details will be handed over to the administrators of the affected networks. This will avoid issues like blocking traffic from legitimate IP addresses due to spoofing of their IP addresses by the scanners. This kind of Denial of Service (DoS) can theoretically be prevented by a white-list, but it requires a substantial administrative overhead to maintain.

- The amount of network data captured or stored for consumption by the detector must be substantially smaller than the original.

Considering the above operating conditions the detector characteristics can be obtained

- It operates on packet level summaries.
- It operates in real time as it has access to all the required packet summaries immediately. Flow level data can only be obtained when the flow is finished and the information is purged to storage. This can take a long time as the flows duration varies greatly. This forces any detector based on flow level data to be non real time.
- It is stateless in nature. Inspecting application level data requires the storage of complete packets and their reassembly requiring the detector to maintain state. We do not require storage or reassembly of packets as we just need the summaries. We can see that the storage requirements for these summaries is based on the volume of packets. A way to decouple the storage requirements with the traffic volume is to use *aggregation*.

3.2 Related work

Network Security Monitor (NSM) [11] was the pioneering NIDS. Its scan detection rules detected any source IP address which attempted to connect more than 15 hosts. Time is not mentioned as a factor in the paper. Since then, most NIDS use a variant of this thresholding algorithm.

Snort has a preprocessor for detecting port scans based on invalid flag combinations or exceeding a preset threshold. Scans which abuse the TCP protocol like

NULL scans, Xmas tree scans and SYN-FIN scans can be detected by their invalid TCP flag combinations. Scans which use valid flags can be detected by a threshold mechanism. Snort is configured by default to generate an alarm only if it detects a single host sending SYN packets to four different ports in less than three seconds [19].

Bro also uses thresholding to detect scans [17]. A single source attempting to contact multiple destination IP addresses is considered a scanner if the number of destinations exceeds a preset threshold. A vertical scan is flagged by a single source contacting more than the threshold number of destination ports. Paxson indicates that this method generates false positives, such as a single source client contacting multiple internal web servers. To reduce the number of false positives, Bro uses packet and payload information for application level analysis.

Staniford et al use simulated annealing to detect stealthy and distributed port scans [22]. Packets are initially pre-processed by Spade which flags packets as normal or anomalous. Spice uses the packets flagged as anomalous and places them in a graph, with connections formed using simulated annealing. Packets which are most similar to each other are grouped together. This approach is used in the detection of port scans.

Threshold Random Walk (TRW) developed by Jung et al requires information if a particular host and service are available on the target network [12]. This information is obtained by analysis of return traffic or through an oracle. They apply sequential hypothesis testing on new connection requests that arrive to determine whether a source is performing a scan. The assumption is that a destination is more likely to respond with a SYN-ACK to a benign source (legitimate connection requests are generally from clients who are aware of the services that exist on the destination), than to a scanner source.

Kompella et al focus on scalable attack detection by aggregating the per flow

state into a data structure they call a *Partial Completion Filter* (PCF) [13]. The PCF data structure is similar to a *counting* Bloom filter [3][4]. State can be evicted from the PCF unlike Bloom filters where this is not possible.

Chapter 4

APPROACH

4.1 Simulation environment

We selected OMNeT++ [25][24]as the simulation environment. OMNeT++ is a discrete event simulator with support for network simulation using the INET framework [23].

There is a distinct separation of form/structure and function/behavior in the OMNeT++ simulator. Simulations are made up of *modules*. There are two types of modules: *simple* and *compound*. A simple module is comprised of its structure (defined in the NED programming language) which is nothing but a container with *gates* or connections with which it communicates with other modules. The behavior of a simple module is defined by its C++ implementation .

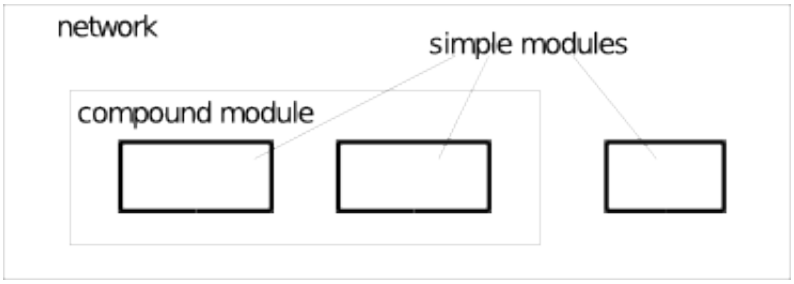

Figure 4.1: Simple and compound modules

4.2 TCP Scanner

TCP has a very complex state diagram (see Figure 4.2 on page 13). The set up of a TCP connection requires a 3-way handshake. The listening application is informed only when the handshake is successful [8].

There are several types of TCP scanning methods used in the field [8]

- TCP connect() scanning
- TCP SYN (half open) scanning
- TCP FIN (stealth) scanning
- Xmas and Null scans
- ACK and Window scans
- RST scans

A TCP connect() scan completes the 3-way handshake and is logged as a connection attempt by the application. This scan is the easy to implement and does not require *root* priviliges. The port is considered open when the connection is established and closed if the connection attempt fails. The scanner sends a SYN packet, receives a SYN-ACK to acknowledge the connection, followed by an ACK by the scanner to complete the connection setup. The connection is then torn down by a FIN from the scanner. This method is only used in port scanning when the scan is run as a normal user. The more typical usage is to probe the application level service version as part of a vulnerability scan.

A TCP SYN (half open) scan is the most popular type of port scan when *root* priviliges are possible. The scan does not show up in the application level logs since

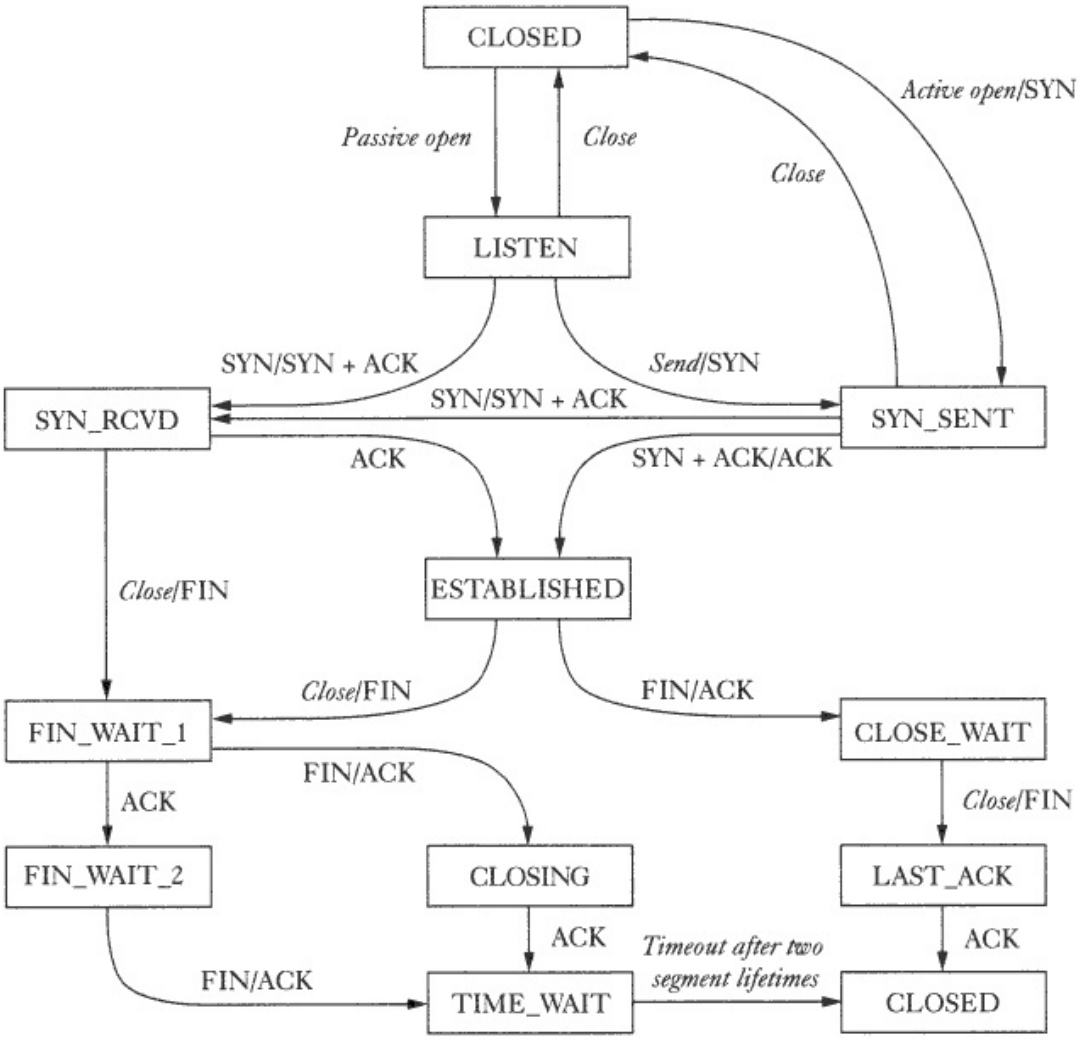

Figure 4.2: TCP State diagram [20]

the 3-way TCP handshake is not completed. Its stops the TCP connection open process midway after the first reponse from the server so is know as the *half open* scan. The scanner sends a SYN packet to the target. If the response is a SYN-ACK, the port is open. A closed port causes the target OS to respond with a RST-ACK. If the reponse received was SYN-ACK, the scanner responds with a RST to abort the connection. The advantage of this method is that scan leaves no trace in the application level service logs.

If there is no response from the target port, the port could be filtered, which means that a firewall is dropping all SYN-ACK packets to the closed port. If that is the case then the FIN scan can be used. The firewall rule set will generally allow all inbound packets with a FIN to pass through without exception. When the scanner sends a FIN packet to a closed port, then the response will be a RST. If the port is open then there will be no response received.

There are several variations of the FIN scan. In a Xmas scan, the URG, PSH and FIN flags are set. In a Null scan, none of the flags are set. In both cases the sequence number is 0.

ACK scans are used to determine which ports are filtered by the firewall by sending a packet to a port with only the ACK flag set. A RST reponse indicates that the port is unfiltered and is accessible remotely. If no response is received or if an ICMP unreachable response is recieved then the port is filtered by the firewall.

We implemented a distributed TCP port scanner in the OMNet++ simulation environment. The scanner supports the TCP SYN (half open) type of scan. The algorithm of the scanner is shown in 4.1.

Algorithm 4.1 TCP scanner

```
Input: Number of scanners n
Input: List of IP/port pairs P
  for every scanner
    portsPerScanner = |P|/|n|
    while portsPerScanner > 0 do
      send SYN
      if recv(SYN+ACK) then
        port OPEN
        send RST
      end if
      if recv(SYN+RST) then
        port CLOSED
      end if
      if recv(TIMEOUT) then
        port FILTERED
      end if
      portsPerScanner = portsPerScanner - 1
    end while
```

4.3 Packet sniffer

Specific packet fields serve as an input to the IDS for generation of the packet summary information. We require the following fields from every incoming IP packet on all the router interfaces..

(1) Source IP (SIP)

(2) Destination IP (DIP)

(3) Source Port (SP)

(4) Destination Port (DP)

(5) SYN

(6) FIN

(7) ACK

(8) RST

We can extract the SIP and the DIP from the IP packet header (see Figure 4.3 on page 17). The other fields are from the encapsulated TCP packet header (see Figure 4.4 on page 18).

Type	Range	Field	Abv.	Extracted from
IPaddress	0.0.0.0 - 255.255.255.255	Source IP	SIP	IP
IPaddress	0.0.0.0 - 255.255.255.255	Destination IP	DIP	IP
Numeric	0 - 65535	Source Port	SP	TCP
Numeric	0 - 65535	Destination Port	DP	TCP
Flag	boolean	Synchronize	SYN	TCP
Flag	boolean	Acknowledgement	ACK	TCP
Flag	boolean	Finish	FIN	TCP
Flag	boolean	Reset	RST	TCP

Table 4.1: Fields extracted by the packet sniffer

The packet sniffer is notified whenever there is an incoming packet on any interface. It is programmed only to extract the required header fields (see Table 4.1 on page 16) even though the sniffer has complete access to the packet header and payload information (sniffer operates in *priviliged* or *root* mode, which allows it to hook into the Operating System (OS) TCP/IP stack).

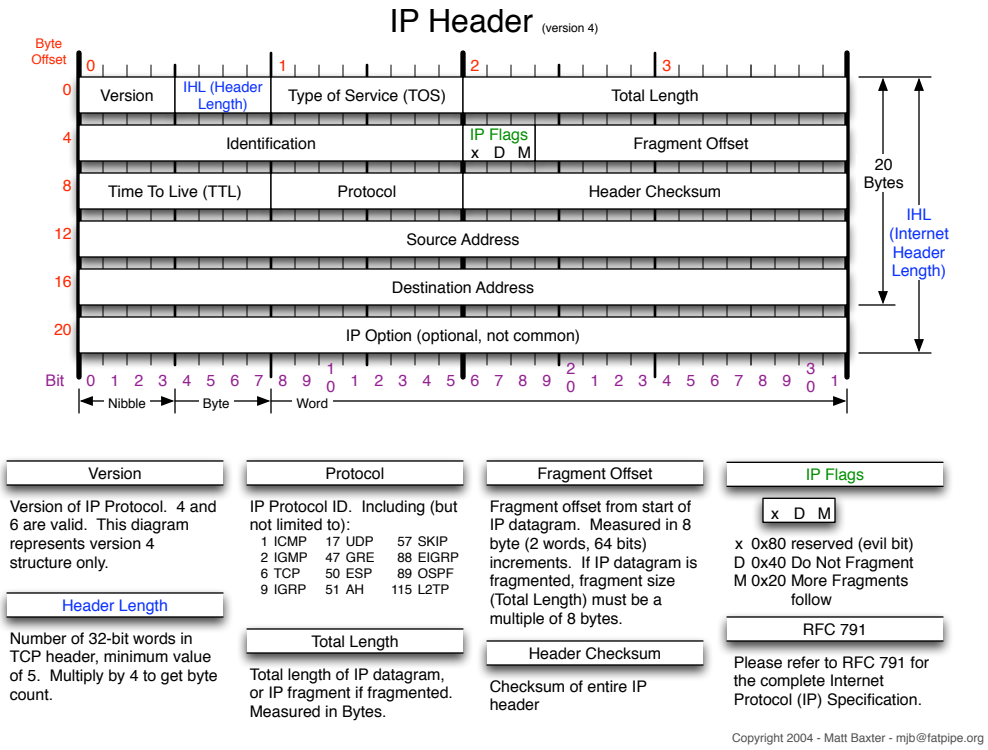

Figure 4.3: IP v4 header [2]

The TCP information is encapsulated within the IPv4 payload. We just peek at the required fields by making a temporary copy of the original IPv4 packet and de-encapsulating it to extract the required TCP fields. The fields are then converted to a text format ready to be pushed to detector mechanism.

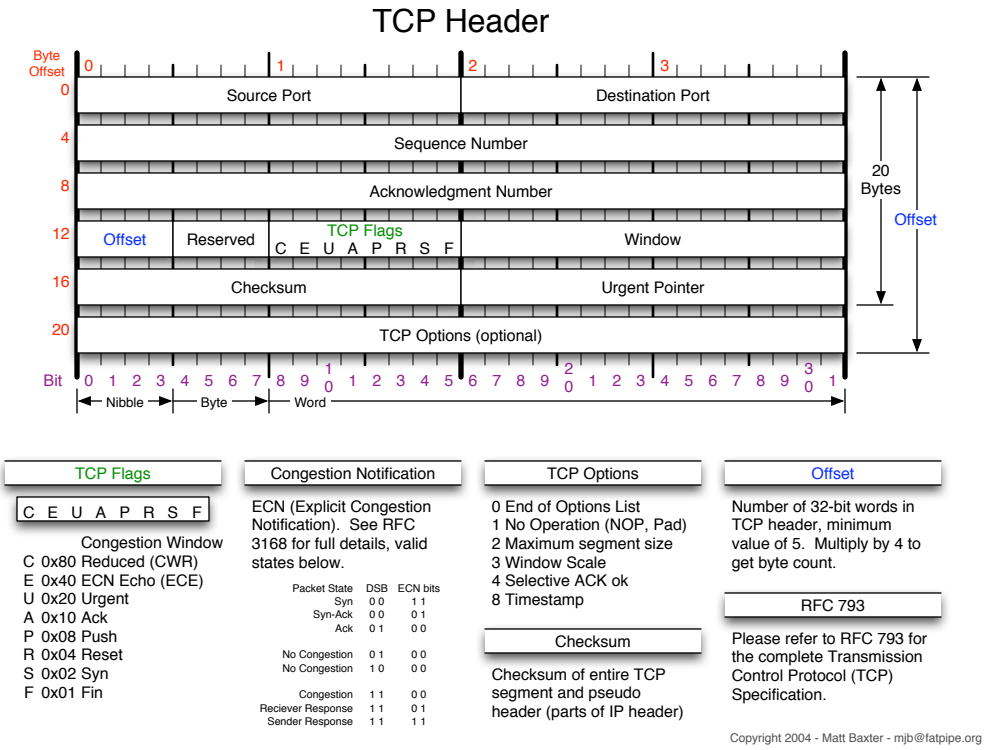

Figure 4.4: TCP header [2]

4.4 Detector

The detector is designed to be strapped on to router firmware. This design choice dictates that the detector must have the following characteristics:

(1) Should NOT be processor intensive.

(2) Very low and predictable memory requirements.

In other words the prime function of a router is packet forwarding and any included Intrusion Detection Sytem (IDS) functionality should scale gracefully and not cause the primary functionality to fail. The emphasis is on realtime detection which means

that processing speed is one of the design goals. We are willing to sacrifice accuracy to some extent to achieve this goal.

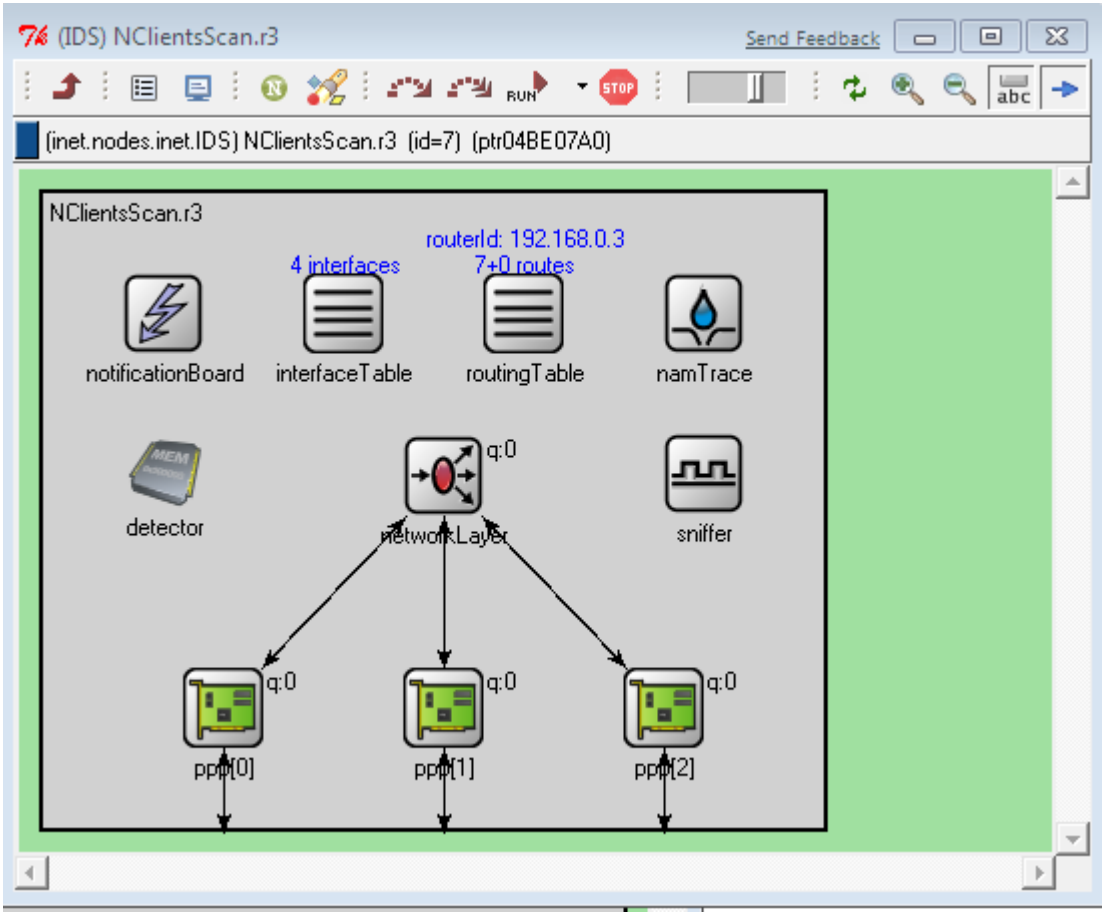

Figure 4.5: Prototype IDS within router r3

The IDS integrated within a router is shown in Figure 4.5 on page 19. The packet sniffer and the detector can be seen in the router. Whenever a packet arrives on a router interface, a lookup of the routing table is performed to determine the next hop if the destination is not local. After the route lookup, the TTL is decremented and the packet is forwarded on the corresponding interface for the particular route.

4.4.1 Patterns in TCP packet traffic

The pattern of benign and TCP scan traffic is different. Our scan detection algorithm uses these differences to flag a particular set of packets as scanners or benign

Symmetry in benign TCP connections

TCP has an elaborate setup and a teardown process. A benign connection will look like the following to an observer of the communication between the client and the server:

$$TCP_{(SETUP)} \Uparrow \overleftrightarrow{Session\ Established} \Downarrow TCP_{(TEARDOWN)}$$

We can see that there are three different stages:

(1) Setup: This is the TCP 3-way handshake.

  (a) SYN

  (b) SYN-ACK

  (c) ACK

(2) Session Established: The period during which the client will communicate with the server. An example would be to fetch a page from a web server.

(3) Teardown: This is when the FIN packet is used to bring down the connection

Asymmetry in TCP scan traffic

We take the TCP SYN (*half open*) scanning into consideration. The traffic between an scanner and a server will look like the following to an observer who is in a position to observer both sides of the communication.

$$TCP_{(OPEN)} \Uparrow \overrightarrow{Handshake\ Aborted} \Downarrow TCP_{(ABORT)}$$

(1) Open: This is the standard TCP 3-way handshake till 1b. Then in 1c the scanner aborts the handshake.

   (a) SYN

   (b) SYN-ACK

   (c) RST

(2) Handshake Aborted: The session was not able to setup as the RST from the scanner aborted the TCP 3-way handshake

(3) Abort: This is when the RST packet aborts the handshake. There is no FIN packet associated with the abort process.

4.4.2 Partial Completion Filter (PCF)

The Partial Completion Filter (PCF) was introduced by Kompella et al. [13]. It is similar to a *counting* Bloom filter. There are multiple parallel stages in a PCF with each stage containing hash buckets that hold a counter (see Figure 4.6 on page 22). The hash bucket counter in scope is incremented for a SYN and decremented for a FIN. For benign TCP connections the symmetry between the SYNs and FINs will ensure that the counter will tend towards 0. If an IP address hashes into buckets which have large counter values in all stages, then we can assert with a high degree of confidence that the IP address is involved in a scan.

4.5 Network Topology

The prototype IDS is deployed on a /16 CIDR [26] within the OMNeT++ simulator. The number of scanners and target servers are variable. There is also a provision to add other hosts which can generate background traffic.

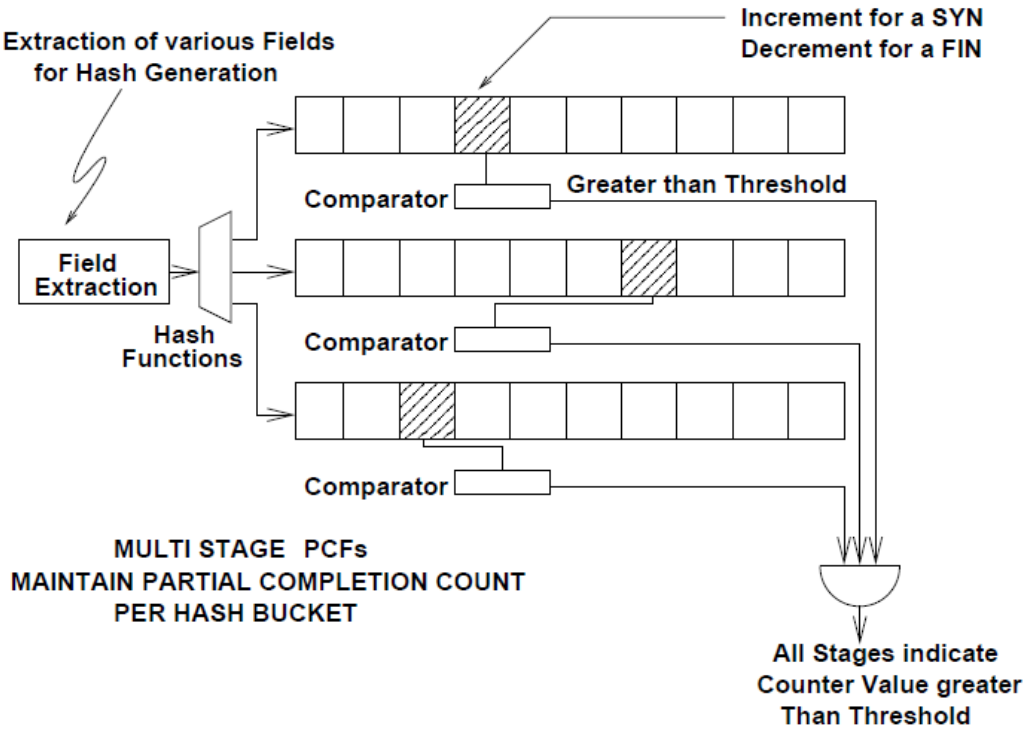

Figure 4.6: Multiple stage Partial Completion Filter [13]

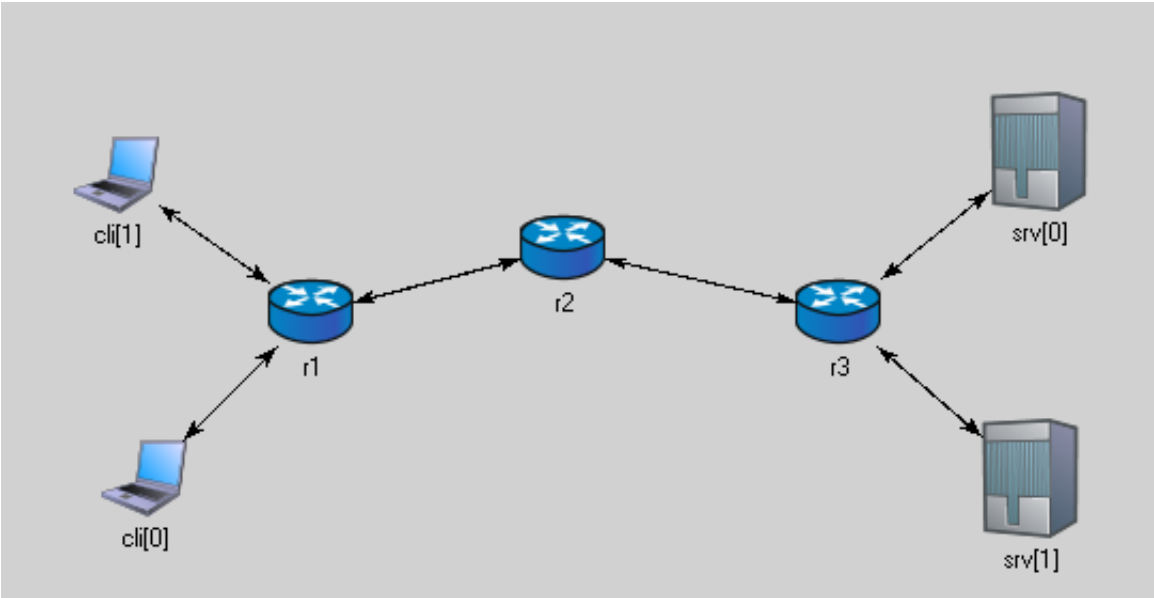

Figure 4.7: Port scan experimental setup in OMNeT++ with 2 scanners, 2 targets, and no background traffic.

Chapter 5

RESULTS

We used an experimental setup with the following configuration.

- Two scanners, two regular routers, one router with the IDS system, and two targets (Figure 5.1 on page 24). The threshold chosen was 3.

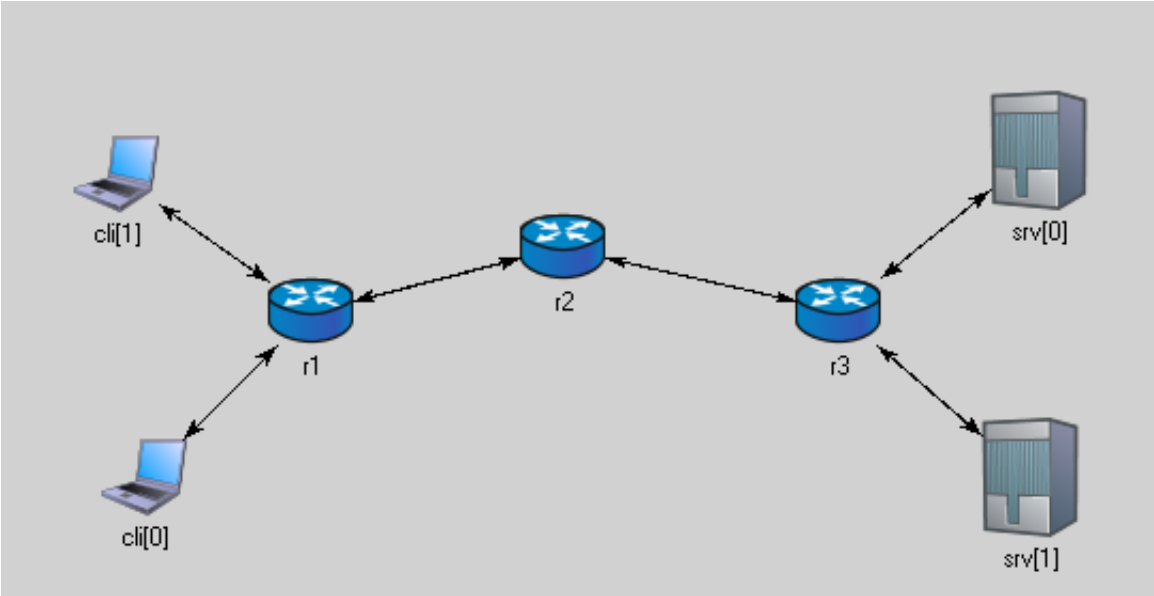

Figure 5.1: Experiment setup with two scanners and two targets

The results are shown in Table 5.1 on page 25.

- Four scanners, two regular routers, one router with the IDS system, and two targets (Figure 5.2 on page 26). The threshold chosen was 3.

Ports	PCF Stages	Buckets/PCF stage	Bucket size	Memory for PCF	Threshold	Detection Rate
4	1	3	32 bits (4bytes)	0.012 kb	3	>90%
10	1	3	32 bits (4bytes)	0.012 kb	3	>90%
20	1	3	32 bits (4bytes)	0.012 kb	3	>90%
4	3	1000	32 bits (4bytes)	12 kb	3	>90%
10	3	1000	32 bits (4bytes)	12 kb	3	>90%
20	3	1000	32 bits (4bytes)	12 kb	3	>90%
4	1	3	32 bits (4bytes)	0.012 kb	1	>90%
10	1	3	32 bits (4bytes)	0.012 kb	1	>90%
20	1	3	32 bits (4bytes)	0.012 kb	1	>90%
4	3	1000	32 bits (4bytes)	12 kb	2	>90%
10	3	1000	32 bits (4bytes)	12 kb	2	>90%
20	3	1000	32 bits (4bytes)	12 kb	2	>90%

Table 5.1: Results of two scanners and two targets

The results are shown in Table 5.2 on page 27.

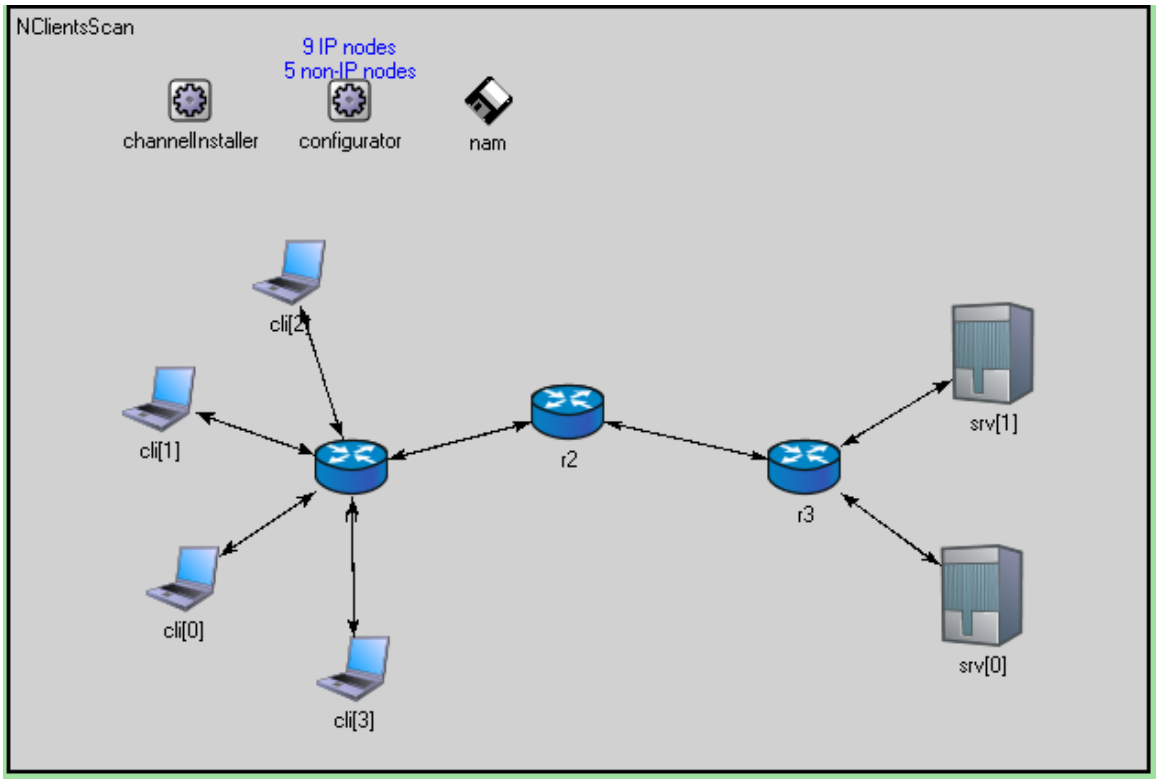

Figure 5.2: Experiment setup with four scanners and two targets

We measure the detection rate as the number of scanner IPs that the detector could identify. The results of both these setups are unusual in that they are constant for a wide variation of parameters. The only parameter which has a significant effect is the threshold. Any scanner that operates below the currently set threshold is mislabelled. Since the amount of traffic generated in the network is limited, it remains to be seen whether this behavior manifests itself in scaled up simulations or actual network traces.

Ports	PCF Stages	Buckets/PCF stage	Bucket size	Memory for PCF	Threshold	Detection Rate
4	1	3	32 bits (4bytes)	0.012 kb	3	>90%
10	1	3	32 bits (4bytes)	0.012 kb	3	>90%
20	1	3	32 bits (4bytes)	0.012 kb	3	>90%
4	3	1000	32 bits (4bytes)	12 kb	3	>90%
10	3	1000	32 bits (4bytes)	12 kb	3	>90%
20	3	1000	32 bits (4bytes)	12 kb	3	>90%
4	1	3	32 bits (4bytes)	0.012 kb	1	>90%
10	1	3	32 bits (4bytes)	0.012 kb	1	>90%
20	1	3	32 bits (4bytes)	0.012 kb	1	>90%
4	3	1000	32 bits (4bytes)	12 kb	2	>90%
10	3	1000	32 bits (4bytes)	12 kb	2	>90%
20	3	1000	32 bits (4bytes)	12 kb	2	>90%

Table 5.2: Results of four scanners and two targets

Chapter 6

CONCLUSION

Conventional Network Intrusion Detection Systems (NIDS) have heavyweight processing and memory requirements as they maintain per flow state using data structures like linked lists or trees. This is required for some specialized jobs such as Stateful Packet Inspection (SPI) where the network communications between entities are recreated in its entirety to inspect application level data. The downside to this approach is that:

- The NIDS must be in a position to view all inbound and outbound traffic of the protected network
- The NIDS can be overwhelmed by a DDoS attack since most of these try and exhaust the available state of network entities.

For some applications like port scan detection, we do not require to reconstruct the complete network traffic. We can see that the aggregation approach works well, somewhat like a set lookup with a very compact storage mechanism. The data structure is unique in following respects:

(1) The values stored cannot be retreived verbatim or enumerated.

(2) An input value can be tested for prior existance among the set of values stored.

These properties listed above are used in reducing the detector state to a constant value. Since routers are devices with limited memory and processing capabilities, these properties fit in exceedingly well with our requirements of fitting a detection mechanism into them.

Gaming the detector system can be attempted in the forward path by sending spurious client generated FINs. This can be countered by eliminating client FINs from the equation. The spurious FIN technique is not possible in the reverse path as the server would have to terminate the connection.

Future work includes incorporating the detector into multiple routers and formulating a peer to peer or client server distributed detector communication network. A distributed set look up is then possible from any point in the network. So routers in various segments can be queried like a directory to check whether a particular packet was forwarded by them.

NOMENCLATURE

ACK TCP ACKnowledge flag

DIP Destination IP

DP Destination Port

FIN TCP FINish flag

HIDS Host-based Intrusion Detection System

IANA Internet Assigned Numbers Authority

IDS Intrusion Detection System

IP Internet Protocol

NIDS Network Intrusion Detection System

OS Operating System

PCF Partial Completion Filter

SIP Source IP

SP Source Port

SYN TCP SYNchronize flag

TCP Transmission Control Protocol

BIBLIOGRAPHY

[1] Allman. A brief history of scanning. In *ACM Internet Measurement Conference 2007*, 2007.

[2] Matt Baxter. Header drawings. Online. Last accessed, April 2009 at `http://www.fatpipe.org/~mjb/Drawings/`.

[3] B Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, (13):422–426, 1970.

[4] A Broder and M Mitzenmacher. Network applications of bloom filters: A survey. In *Internet Mathematics*, pages 636–646, 2002.

[5] Fred Cohen. Simulating cyber attacks, defenses, and consequences. Online. Last accessed, April 2009 at `http://www.all.net/journal/ntb/simulate/simulate.html`.

[6] Various contributors. Squid: Optimizing web delivery, 2008. Last accessed, March 2008 `http://www.squid-cache.org/`.

[7] Ron Deibert and Rafal Rohozinski. Tracking GhostNet: Investigating a cyber espionage network. Online, March 2009.

[8] fyodor. The art of port scanning. *Phrack Magazine*, 7(51), 1997. Last accessed, January 2009 at `http://www.phrack.com/issues.html?issue=51&id=11`.

[9] C Gates, J McNutt, J Kadane, and M Kellner. Detecting scans at the ISP level. Technical Report CMU/SEI-2006-TR-005, Software Engineering Institute, Carnegie Mellon University Pittsburgh, PA 15213, 2006.

[10] Siobhan Gorman. Electricity grid in U.S. penetrated by spies, 2009. Last accessed, April 2009 at `http://online.wsj.com/article/SB123914805204099085.html`.

[11] L.T. Heberlein, G.V. Dias, K.N. Levitt, B. Mukherjee, J. Wood, and D. Wolber. A network security monitor. pages 296–304, May 1990.

[12] J Jung, V Paxson, A W Berger, and H Balakrishnan. Fast portscan detection using sequential hypothesis testing. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2004.

[13] R R Kompella, S Singh, and G Varghese. On scalable attack detection in the network. In *IMC 04: Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement*, pages 187–200. ACM Press, 2004.

[14] Ofer Maor. Divide and conquer: Real world distributed port scanning. RSA Conference, Feb 2006. Last accessed, March 2008 at `http://www.hacktics.com/frpresentations.html`.

[15] Elinor Mills. Just how vulnerable is the electrical grid?, 2009. Last accessed, April 2009 at `http://news.cnet.com/8301-1009_3-10216702-83.html`.

[16] S Panjwani, S Tan, K Jarrin, and M Cukier. An experimental evaluation to determine if port scans are precursors to an attack. In *Proceedings of the 2005 International Conference on Dependable Systems and Networks*, pages 602–611, 2005.

[17] V Paxson. Bro: a system for detecting network intruders in real-time. *Computer Networks*, (31):23–24, 1999.

[18] Jon Postel. IANA-Internet Assigned Numbers Authority Port Number Assignment. Online. Last accessed, April 2009 at `http://www.iana.org/assignments/port-numbers`.

[19] Martin Roesch. Snort - lightweight intrusion detection for networks. In *LISA '99: Proceedings of the 13th USENIX conference on System administration*, pages 229–238, Berkeley, CA, USA, 1999. USENIX Association.

[20] Shweta Sinha. TCP state transition diagram. Online. Last accessed, April 2009 at `http://www.winlab.rutgers.edu/~hongbol/tcpWeb/tcpTutorialNotes.html`.

[21] Edward Skoudis and Tom Liston. *Counter Hack Reloaded: A Step-by-Step Guide to Computer Attacks and Effective Defenses (2nd Edition)*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2005.

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

[23] A Varga and Others. INET framework for OMNeT++ 4.0, 2009. Last accessed, March 2009 at `http://inet.omnetpp.org/`.

[24] A Varga and Others. OMNeT++, 2009. Last accessed, March 2009 at `http://www.omnetpp.org`.

[25] András Varga and Rudolf Hornig. An overview of the OMNeT++ simulation environment. In *Simutools '08: Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*, pages 1–10, ICST, Brussels, Belgium, Belgium, 2008. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[26] Wikipedia. Classless inter-domain routing — wikipedia, the free encyclopedia, 2009. Last accessed, April 2009 at `http://en.wikipedia.org/w/index.php?title=Classless_Inter-Domain_Routing&oldid=281677018`.