IPv6 Network Security Monitoring: Similarities and Differences from IPv4

Qinwen Hu

A thesis submitted in fulfilment of the requirements for
the degree of Doctor of Philosophy in Computer Science,
The University of Auckland

September 2016

Abstract

Since about 2000, the Internet has become part of our daily lives. IPv4 has been the main protocol used for the current Internet structure. However, the size of the address space in IPv4, which limits the number of available addresses does not meet the growth needs of the Internet. IPv6 is designed to support the accelerated growth of internet enabled applications and devices. Moreover, this new protocol is expected to solve many problems in the existing IPv4 networks, and, most importantly, make the Internet more secure. From the security perspective, IPv6 is similar to IPv4 with larger addresses that may stop attackers from finding a target host using only the traditional reconnaissance technologies. However, such defensive obscurity depends upon how network administrators assign IPv6 addresses. In this study, we consider the trend of how different address allocation mechanisms are performed in last five years and how feasible it is to discover IPv6 hosts from the public DNS servers, by launching two large-scale surveys. Our results show that IPv6 assignment has become more secure compared to the results from the past five years. However, we detected some potential issues in current DNS reverse zone deployment; we provide some recommendations for planning and deploying IPv6 addresses. Furthermore, We discussed what are the key architecture considerations for using open source Intrusion Detection Systems (IDSs) in a high speed network, and how an open source IDS should be designed to more readily facilitate new emerging IPv6 attacks. We designed and proposed a new solution for detecting the IPv6 DNS reconnaissance attack. We demonstrated the feasibility, or otherwise, of implementing this new mechanism in the three IDSs, and we demonstrated strengths and weaknesses of implementing this new detection approach in three IDSs. We suggest that IDS developers should release more IPv6 rules or policies to handle emerging IPv6 threats. All the experimental environments and tools used in this study are explained in the thesis.

Acknowledgments

I would like to express the deepest appreciation to my supervisor, Associate Professor Nevil Brownlee. It has been an honour to be his Ph.D. student. He has been a wonderful mentor for me, giving me invaluable advice, criticism and correction of my research from beginning to end. I appreciate all his contributions in time and ideas to make my study productive and stimulating. I am also very grateful to Professor Brian Carpenter for his scientific advice, knowledge and many insightful suggestions. I would like to thank Dr Giovanni Russello, Dr Ulrich Speidel and Dr Aniket Mahanti. Dr Speidel suggested the use of a frequency distribution plot to examine the discrete probability function among IID values in our conference paper. Dr Russello and Dr Mahanti helped me to review my thesis and conference slides. I am honoured by their help and encouragement throughout my studies.

In regards to our two global IID surveys, I acknowledge the very great help of Tsinghua University, which provided technical support and resources in mainland China. I also recognize the support and help of Dr David Plonka at Akamai Technologies [1]; he shared with me some of the knowledge that Akamai Technologies picked up during its IID allocation survey. The probability of a pseudo-random IID being misclassified as an EUI-64 would not have been corrected without David's suggestions. In regard to my later work on intrusion detection systems, I would also like to thank Russell Fulton at the ITS security team; he made significant contributions to this project. He always provided me with thoughtful information and helped me set up three IDS tools at the beginning of my research.

The members of the network research group have contributed immensely to my study time at the University of Auckland. The group has been a source of friendships as well as of good advice and collaboration. I would like to express my sincere thanks to Se-Young Yu, who helped me set up a high-speed network environment. I very much appreciate his enthusiasm, intensity, and willingness. Even during the most difficult times when configuring the experimental environment, he always came up with some helpful ideas and was very supportive. In addition, I would like to acknowledge Dr Habib Naderi, who studied here a couple of years ago; he has been supportive since I began working on my Master's thesis. He always encouraged me and guided me in overcoming some problems. Other present group members that I have had the pleasure to work with are Lei Qian and Maziar Janbeglou. I also thank everyone at the University of Auckland who has helped

with the project and supported me in many ways: Dr Yu-Cheng Tu, Lixin Yang, Haokun Geng, Marc Jeanmougin, Sean Davidson, Robyn Young, Sithra Sukumaar and Yolande Young. I also thank my friends Lurong Liao and Minghan Wu for providing support and friendship that I needed. I also thank John Moriarty for ensuring correct grammar in my thesis.

Finally, I would like to thank my family for all their love and encouragement. My parents, Mr GuiQiao Hu and Ms Yan Zhou supported me in all my pursuits. Additionally, special thanks to the newest additions to my family, Yue (Bonnie) Wang, my wife, and her wonderful family (Mr Qun Wang and Ms HuaJuan Qi), who have all been supportive and caring.

The best outcome from these past four years of my doctoral study is finding my best friend, soul-mate, and wife, Yue (Bonnie) Wang. Bonnie has been a great support during my good and bad times. These past several years have not been an easy ride for either of us; I truly thank Bonnie for staying by my side, even when I was depressed. I feel that we have strengthened our commitment to each other; we will live life to the fullest.

Declaration

These doctoral studies were conducted under the supervision of Associate Professor Nevil Brownlee. This thesis contains sections from the following previous publications.

"IPv6 Host Address Usage Survey" [2] in Section 3.

"How Interface ID allocation mechanisms are performed in IPv6" [3] in Section 3.

The work presented in this thesis is the result of original research carried out by myself, in collaboration with others, whilst enrolled in the Department of Computer Science as a candidate for the degree of Doctor of Philosophy. This work has not been submitted for any other degree or award in any other university or educational establishment.

Qinwen Hu,
Auckland, New Zealand,
September 2016

Abbreviations

Acronym	Description
6LoWPAN	IPv6 over Low power Wireless Personal Area Networks
ACK	ACKnowledgement
AH	Authentication Header
AKD	Authoritative Key Distributor
API	Application Programming Interface
APPA	Automatic Peer-to-Peer Anti-spoofing
ARP	Address Resolution Protocol
BPF	Berkeley Packet Filter
CGA	Cryptographically Generated Address
CUSUM	Cumulative Sum
DAD	Duplicate Address Detection
DDoS	Distributed Denial of Service
DFA	Deterministic Finite Automaton
DHCPv6	Dynamic Host Configuration Protocol version 6
DNS	Domain Name System
DoS	Denial of Service
DUID	DHCP Unique Identifier
ESP	Encapsulating Security Payload
FDT	Fast Data Transfer
FIN	FINish
FTP	File Transfer Protocol
GUI	Graphical User Interface
HBA	Hash Based Addresses
HIDS	Host Based Intrusion Detection System
HMAC	Hash Message Authentication Code
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol

ICMP	Internet Control Message Protocol
ICV	Integrity Check Value
IDS	Intrusion Detection System
IETF	Internet Engineering Task Force
IID	Interface Identifier
IKEv2	Internet Key Exchange Protocol version 2
IP	Internet Protocol
IPFIX	IP Flow Information Export
IPsec	Internet Protocol Security
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
ISAKMP	Internet Security Association and Key Management Protocol
ISATAP	Intra-Site Automatic Tunnel Addressing Protocol
LAN	Local Area Network
LTA	Local Ticket Agent
MAC	Media Access Control
MD5	Message Digest 5
MIME	Multi-Purpose Internet Mail Extensions
MITM	Man-In-The-Middle
MLP	Multi-layer Perceptron
MPLS	Multiprotocol Label Switching
MSS	Maximum Segment Size
MTU	Maximum Transmission Unit
NA	Neighbour Advertisements
NAT	Network Address Translation
ND	Neighbour Discovery
NFA	Non-deterministic Finite Automaton
NIC	Network Interface Controller
NIDS	Network Intrusion Detection Systems
NN	Nearest Neighbour
OHC	One-way Hash Chain
OS	Operating System
OSI	Open Systems Interconnection
OTC	One-Time-Cookies
OUI	organisationally Unique Identifier
PHF	Portable Hypertext Format
PTR	Pointer Record
RA	Router Advertisements
RAC	Reconnaissance Alter Correlation
RAP	Reconnaissance Activity Profiler

RIDS	Reconnaissance Intrusion Detection System
RIR	Regional Internet Registry
RPC	Remote Procedure Calls
RS	Router Solicitations
RST	ReSeT
RTT	Round Trip Time
RTR	Reliable Transaction Router
SA	Security Association
SHA-256	Secure Hash Algorithm 256
SHA-384	Secure Hash Algorithm 384
SHA-512	Secure Hash Algorithm 512
SLAAC	Stateless Address Autoconfiguration
SMOTE	Synthetic Minority Over Sampling Technique
SMTP	Simple Mail Transfer Protocol
SNMP	Simple Network Management Protocol
SOA	Start of Authority
SQL	Structured Query Language
SSH	Secure Shell
TCP	Transmission Control Protocol
TTL	Time to Live
UDP	User Datagram Protocol
UDT	UDP-based Data Transfer
URI	Uniform Resource Identifiers
VoIP	Voice over IP
VNC	Virtual Network Computing
WWW	World Wide Web
XSS	Cross-Site Scripting

Contents

List of Figures

List of Tables

Chapter 1

Introduction

We live in an increasingly digital world; computer networks in a variety of forms have become part of our everyday lives. The Internet has changed the lives of the majority of people. It has given greater opportunities and more markets than people could dream of before; moreover it has made life easier and accessing media and technology more convenient. But any sufficiently advanced technology always has negative as well as positive aspects. The positive side of computer networks is increased business opportunities. The negative side is that significant security issues have emerged.

In the past, only some industries, such as the finance and manufacturing industries experienced network security issues with their customers. See Figure 1.1 [4]

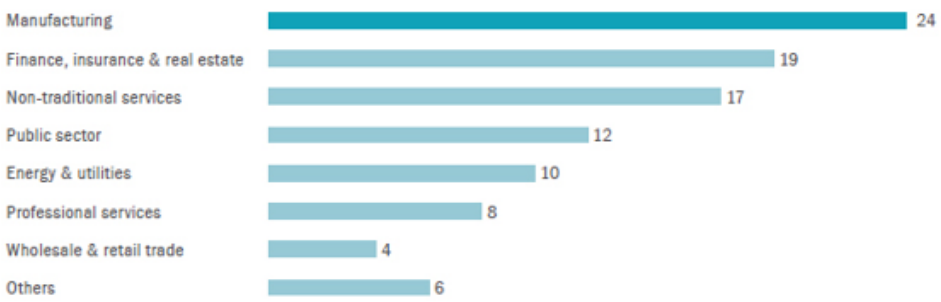

Figure 1.1: Cyber-attacks frequency by Industry in 2012 (Source from [4])

However, as network technologies continue to develop, more and more industries will face security problems. Now, many network users believe that network security has become the top priority of all organisations that want to keep their customer's information safe and protect their system from serious threats. Figure 1.2 demonstrates the Distributed Denial of Service (DDoS) affect for different industries in 2015.

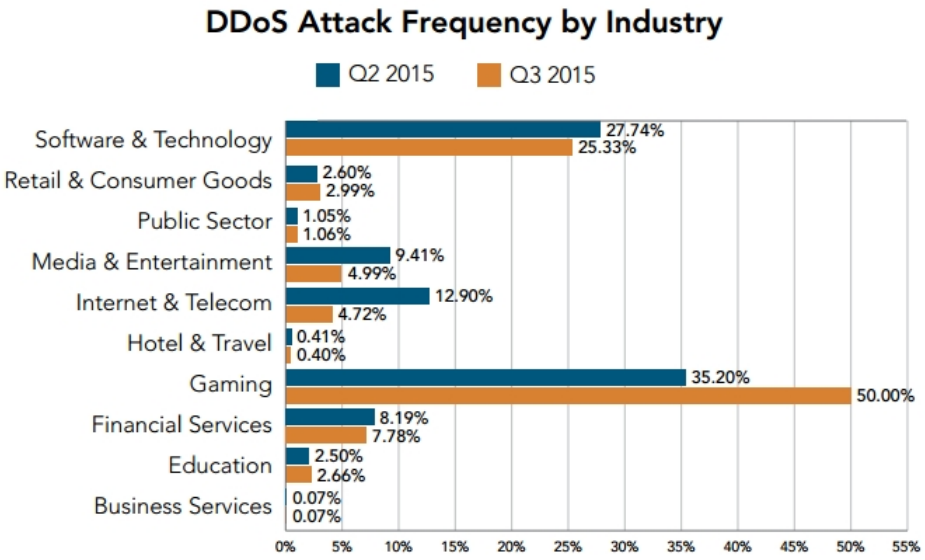

Figure 1.2: Cyber-attacks frequency by industry in 2012 (Source from [5])

Table 1.1: Some common threats and a few specific techniques used for each threat

Threat name	**Techniques**
Reconnaissance	Internet Protocol (IP) sweeps, port scanning
Fraud	Trojans, Phishing [11] and IP address spoofing
Sabotage	Viruses and Denial of Service (DoS) [12] attacks
Spying	Man-In-The-Middle (MITM) [13] and pervasive monitoring [14]

The number of network security issues (See Table 1.1)has continuously risen since the year 2000 (Figure 1.3). Nowadays, system integrity and availability are not the only target of network attacks, instead more and more attacks aim to explore the user information contained within the system. For instance, in 2013, Ninemsn staff [15] reported that an Australian online dating site had been hacked, the attacker harvested 42 million user records including names, email addresses and unencrypted passwords.

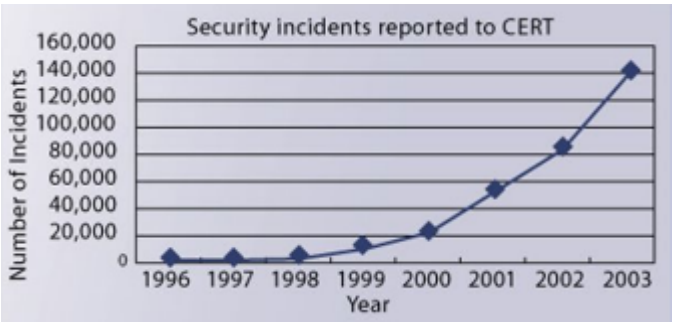

Figure 1.3: Security incidents continue to increase (Source from [6])

In this study, the focus is on the security issues that are exposed by the malicious launching of attacks via the network infrastructure. A timeline is used to demonstrate

the evolution of attacks that indicate where IPv6 introduces new issues. Moreover the research questions and thesis structure are presented later in this chapter.

The remainder of this chapter is organised as follows: Section 1.1 explains why the security attacks are increasing dramatically, what causes the attacks, and how network security can be improved. Section 1.2 describes the objective of this study, explains why this is important. The following Section (1.3) shows the area of this study and the related research question are discussed in Section 1.4. Section 1.5 highlights the contribution from this study, and the overview of the thesis structure is covered in Section 1.6.

1.1 Where are we in terms of network security?

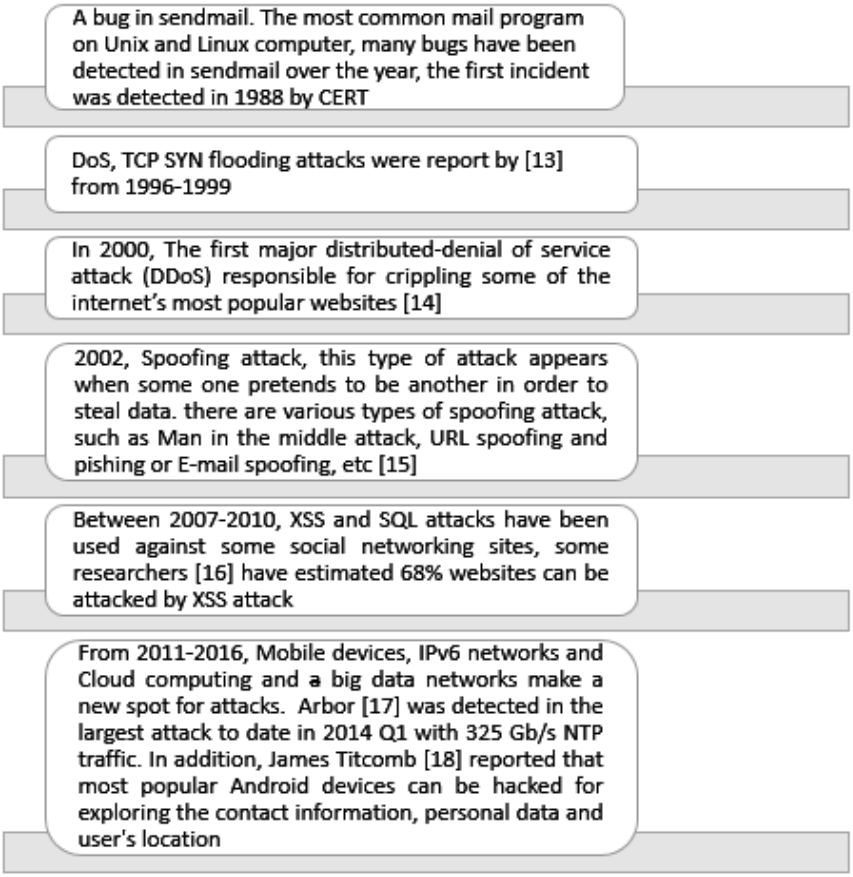

Figure 1.4: Timeline of the evolution of attacks

Based on annual security reports from McAfee [16] and Symantec [17] for past ten years, we have identified the following changes:

- Network threats are increasing dramatically: business [18] and government [19] users have been plagued by a series of network attacks. Moreover, some small and medium businesses and individuals have been targeted by compromising a system user who has only limited access to network or administrative resources. A compromised user can give attackers a beachhead into an organisation from which to mount additional

attacks on the enterprise from the internal network. Even though multiple solutions are used to defend against attacks, still attackers find new and different methods to evade them

- Network threats are more intricate: nowadays, the security attacks have become benefit-driven ([11], [20]), the network threats changed from simple ones to distributed, cooperative and intricate attacks, such as DDoS, worm and DNS amplified attacks [21]. Figure 1.3 demonstrates the steps towards today's attack trends
- Standardized equipment makes network attacks easier: before the 1990s programmers designed specific system applications and the network environment for an individual company. Therefore each company had a different network and operating system, and if, for instance, you wanted to attack one network, you had to understand its environment. Today most companies use the same hardware components, with the same or similar operating systems: the only thing that attackers need to know is the most common systems, such as UNIX, Microsoft, IOS, Android and Cisco, then they can break into most systems around the world
- Some security tools are using an ineffective approach: some security tools are not efficient for real-time attacks, because these tools are using a "threat-based security" approach, which needs new worms or attacks to happen, and only then can they detect or solve future instances of the problem by updating the security database or policies. However, the damage is already done and the new solution cannot recover a file which has been modified or deleted

To summarise the general trend of current network security, the protection task is becoming harder and harder; users are facing an "arms race" scenario between attackers and system administrators. The existing security tools cannot guarantee to detect and solve every network attack, but these techniques can be used to reduce risk. Some common security tools are:

A firewall allows users to configure forwarding policies for dropping unmatched traffic. However, some previous studies [22, 23, 24] show that attackers can easily create malicious traffic to pass through the firewall checking. For example, when a firewall is configured to only forward a packet that contains a particular Transmission Control Protocol (TCP) port, when this firewall is presented with a fragmented packet, the first fragment does not contain the port information. In this case, the firewall may fail to enforce its forwarding policy; the firewall allows the subsequent fragments to pass without any checking.

Antivirus programs that aim to detect and defend your computer against attack by malicious programs. This solution uses signature based detection; each signature contains special patterns or behaviours that have been learned from previous attacks. Antivirus programs are useful to detect already known attacks or their slight variations, but not the new ones or malicious variations that defeat the pattern recognition engine.

Encryption software that uses digital signatures or security certificates to protect data confidentiality and integrity, so that the encrypted data cannot be read or modified with-

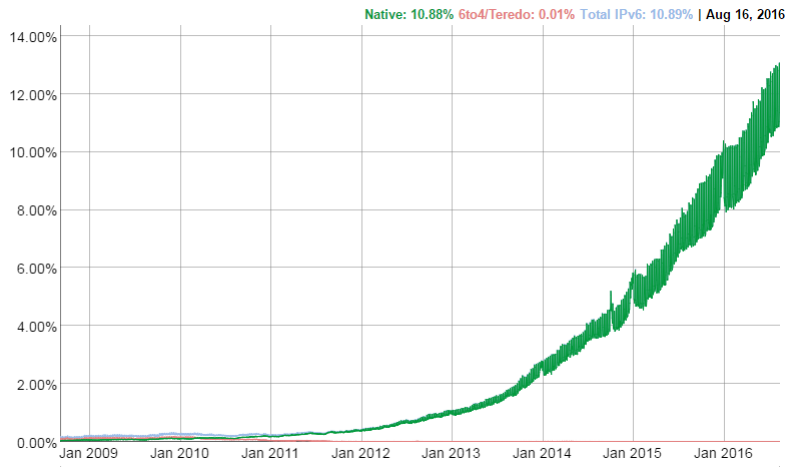

Figure 1.5: The percentage of users that access Google over IPv6 (Source from [7])

out the correct keys or certificates.

Authentication is the process to confirm identity or ownership. For instance, an online bank can confirm a person's ownership of the data by validating their username and password.

Intrusion detection systems that analyze network activities to detect malicious actions and behaviours that can compromise the security of a network host; they use default or user defined rules that try to find malicious behaviours from incoming and outgoing packets. If any packet matches a defined rule, the system either logs the incidents or sends the alarm to the administrators based on the system configuration.

1.2 Objective

Standardised in 1998, IPv6 is a new protocol for the future network. We observe that IPv6 has been deployed more rapidly since 2009. Figure 1.5 shows that the number of IPv6 users increases every year. Therefore, we believe that analyzing network behaviours in IPv6 networks will yield significant results, which will help us to better understand the potential network security issues in IPv6 networks.

1.3 Areas of study

There are still some issues in IPv6 protocol deployment to examine and also some areas in which to research new security issues in IPv6. First, a brief description of issues is presented here. Some existing IPv4 security problems will extend to the IPv6 environment, and some common attacks will still occur in the IPv6 network, such as XSS, SQL injection, because these kind of attacks happen in the application layer, the issues will still be present when change happens in the network layer only. Normally, the application layer attacks will gather system information and exploit existing software bugs or crash the system by attacking system vulnerabilities. This is usually done by scanning the system information, searching the system vulnerability information, then sending special packets to trigger system crashes or to access the system via some existing ports then installing

back doors. Then they can easily access the system at some later date. For example, a backdoor attack is a typical logical threat, which lets attackers control a user's computer via the Internet without their permission. Attackers normally hide the backdoor Trojan as legitimate software or a link in spam E-mail. When the user runs the Trojan, it will automatically add itself to the computer startup register, so the attackers can monitor and control the computer whenever the user is using the Internet.

Immature IPv6 products: some firewalls still do not know how to handle IPv6 extension headers correctly [25]. That leaves two choices for network administrators: drop all IPv6 packets that contain an unrecognised extension header type, or allow packets containing unknown extension headers. The former choice is an undesirable compromise in service terms, while the latter creates a security concern.

IPv6 vulnerabilities: IPv6 brings in some new features, such as the big address space that reduces the address exhaustion issue, the address allocation becomes easier and simpler through the stateless autoconfiguration mechanism, but we need to deal with some security concerns when we want use these features. For example, in IPv6 the stateless auto-configuration approach, the EUI-64 address allocation mechanism exposes user activities by using the constant Interface Identifier field when moving from one subnet to other subnets. Moreover, the stateless auto-configuration approach may introduce other issues, for instance:

Address accountability: IPv6 is similar to IPv4 with larger addresses. From the security point of view, there is no significant difference between IPv4 and IPv6. In some scenarios, IPv6 is slightly more secure, technologies such as CGA can be used for reducing the address spoofing attacks, the privacy IPv6 address allocation mechanism makes the traditional address scanning attack less feasible. However, many studies [26, 27, 28] have observed that there are numerous ways for attackers to reduce address scanning space by using common patterns in the Interface Identifier (IID) field. For example the EUI-64 allocation mechanism introduces two common patterns that cut down the search space in the IID field. More detail is given in Section 3.2.1.1.

Duplicate Address Detection (DAD) flooding attack: a new machine may send a DAD request to check whether a address is already used or not. If attackers on the same link respond to all the DAD checks by saying "I own this address", this will stop the new machine from getting a globe IPv6 address and generate a new DoS attack in the IPv6 network. More detail can be found in Section 2.4.1.2.

Router advertisement spoofing attack: the attackers can spoof the router advertisement messages to modify the routing table at a victim PC, which could redirect all traffic to pass to a host controlled by the intruder. More detail in the Section 2.4.1.3.

Leveraging DNS Reverse Zone: because traditional address probing had less success in IPv6 networks, attackers started to search for new ways to gain host address information. Chown et al. [28] believe the DNS server will become a new target for attackers wishing to explore IPv6 addresses. For instance, Van Dijk [26] discovered that DNS reverse mappings

can be used for discovering IPv6 nodes. He disclosed that the attacker only needs to walk through the target 'ip6.arpa' zone by issuing queries for Pointer Record (PTR) records corresponding to the domain name. More detail can be found in section 3.1.3.

Neighbour Discovery (ND) cache exhaustion: In [29], Gashinsky et al. discussed a potential denial of service condition against the router that performs address resolution on a large number of destination addresses. In IPv6, the subnets are quite big, most existing devices only save assigned addresses. The ND cache table can be prematurely exhausted by sending a larger number of Internet Control Message Protocol version 6 (ICMPv6) NA responses. More detail of ND cache exhaustion can be found in Section 2.3.1.5.

Co-existence/transition mechanisms vulnerabilities: the transition solutions are currently part of the migration plans for using IPv4 and IPv6 environments in parallel, which allows two IPv6 machines to communicate over an IPv4-only network. However, recently some issues have been found in these proposed solutions [30, 31], such as how to prevent a DoS attack on a stateful translation gateway, or how to reduce memory usage by maintaining double routing tables for dual stack solutions. More detail of transition mechanisms vulnerabilities can be found in 2.3.4.

As we mentioned above, many studies [32, 33] have researched what potential issues will appear in the IPv6 network. However, most of these studies believe that the reconnaissance attack will be less feasible in the IPv6 network because of the 128-bit address space. For instance, they claim that it is infeasible to scan a /64 network in a short period. In [24], Chown suggests that if an attack sends a probe per second, it will take 500,000 years to complete a search of a 2^{64} address space. However, while we believe the old address scanning techniques may be less successful in IPv6, that does not mean no new techniques will be created. Nowadays, the amount of malicious traffic is increasing and new attacks are observed every year. Many new IPv6 attacks have been observed in previous studies [29, 34, 35], so it is reasonable to do research that compares the similarities and differences between IPv4 and IPv6 attacks, in order to assist in the identification of new security issues in IPv6 networks. Understanding these vulnerabilities will help in planning and deploying an IPv6 network successfully. Furthermore, this research is intended to identify which security tool could be chosen to most effectively detect malware in its earliest phase. There are some existing solutions on the market, such as firewalls that can filter inbound network traffic, Antivirus software used to stop worms and similar malware, authentication requirements that introduce an access control mechanism, and VPNs that encrypt dataflow between headquarters and agencies over the Internet. However, some mechanisms have limitations in detecting attacks from the internal network, while others cannot inspect the contents of data packets. It is important to put in a second, but not secondary, line of defence: an IDS which helps network administrators to detect unwanted traffic passing through a network or being forwarded to a particular device. The IDS can be applied at either software or hardware level. It monitors and

detects intrusion activities or events, for instance illegal and malicious traffic or traffic that violates a security policy.

1.4 Research questions

After the above brief explanation of our motivation, we raise the following questions:

Q1: What are the main differences between IPv4 and IPv6 attacks? What kinds of new attacks may be seen?

Q2: How will reconnaissance attacks change in IPv6 networks?

Q3: What are the key architectural considerations for open source IDSs as network operating speeds increase?

Q4: How should an open source IDS be designed to more readily facilitate the interception of newly emerging IPv6 attacks?

1.5 Contributions

From this study come several scientific contributions that have not been investigated in the real IPv6 network environment before, but may help users to better understand the potential security vulnerabilities of existing IPv6 deployments. In addition, the strengths and weaknesses of using IDS solutions in a high speed network and IPv6 environments are demonstrated. It is very important to understand these weaknesses, as that can help in planning and deploying an IPv6 network successfully. This study:

1. Provides detailed analysis on whether network administrators were aware of the risk of IPv6 reconnaissance attacks. There have been many studies that have mentioned DNS reconnaissance attacks [26, 28], but this study demonstrated by launching a survey crossing five regions and fifty countries that DNS reconnaissance is still effective in IPv6. Moreover, it presented the trend of address allocation mechanisms usage during the last five years (2009-2014); the trend shows that IPv6 client addresses are increasingly being assigned by a randomized IID allocation mechanism, as network administrators become aware of the need to use non-predictable patterns for IPv6 clients. Furthermore, existing tools (Metasploit, Nmap or Nessus) were tested in use to simulate new IPv6 attacks in an experimental environment; most such tools (Nmap or Metasploit) were not able to launch new IPv6 attacks. However, some new tools have been introduced, such as
    - THC-IPv6: The toolkit provides many programs to access the IPv6 vulnerabilities, such as fakerouter6 which can fake the RA response and announce itself as a router on a network. Again the floodroute6 will send random RAs to flood the target device. The parasite6 can launch a MITM by sending ICMPv6 neighbor solitication/advertisements
    - SI6 Networks' IPv6 Toolkit: This package aims to help both IPv6 vendors improve the security and resiliency of their products, for instance: frag6 provides

accessing IPv6 fragmentation and reassembly function, it can be used for either flooding a target or accessing the fragment reassembly policy of a given node

- Linux Kali: this is a package that contains a large number of penetration testing tools. This toolkit provides various test cases for IPv4 and IPv6

2. Snort, Bro and Suricata were shown to have good support for detecting the most common attacks in IPv6 environment, such as port scanning, TCP SYN scanning, etc. However, none of them detect the IPv6 DNS reconnaissance attack or new IPv6 attacks. In this thesis, a new solution is designed and proposed for detecting IPv6 DNS reconnaissance attacks. The feasibility of implementing this new mechanism in the three IDSs was investigated and commented upon. Additional information is provided on the limitations of implementing the new mechanism in Snort.
3. We have summarised similarities and differences between IPv4 and IPv6 attacks, by comparing the existing attacks and new attacks, it was observed that most attacks share some common strategies. In particular a reconnaissance attack is an initial step toward finding hosts or system vulnerabilities. Spoofing attacks have been detected in both IPv4 and IPv6 as well as DoS attacks. Furthermore, IPv6 was found to be structurally similar to IPv4 but with larger addresses. From the security point of view, there are no significant difference between IPv4 and IPv6. In some scenarios, IPv6 is slightly more secure, technologies such as Cryptographically Generated Addresses (CGA) can be used to reduce the address spoofing attacks, the privacy IPv6 address allocation mechanism makes the address reconnaissance attack less feasible. However, in other cases IPv6 introduces new attacks, for instance the extension header attack, the ND cache exhaustion attack and the ND spoofing attack.
4. The current study provides a thorough investigation on the behaviour of launching each IDS on a high speed network. This provides deep understanding on whether the existing features are able to handle 10 Gb/s network traffic, and how they behave in various network circumstances. The investigation demonstrated the strengths and weaknesses of different configurations on a high speed network and provided indicators of how future IDSs should be designed to be compatible with the high speed network.

1.6 Structure of thesis

Chapter 1 provides brief background information on security issues. The basic security concepts are explained with an introduction to existing security issues and a discussion of some common solutions for preventing network attacks. In addition, the motivation for and contribution by this study are discussed in this chapter.

Chapter 2 answers research question one by comparing the existing IPv4 attacks and new IPv6 attacks, it is noted that IPv6 is similar to IPv4 with larger addresses. Some common security issues in IPv4 networks are reviewed. Next, new security issues within IPv6 are discussed followed by some new tools are used for launching new IPv6 attacks.

Moreover, it is shown why a reconnaissance attack is the first phase of launching a successful attack, and why the traditional reconnaissance attack becomes less feasible, if the IPv6 host uses a randomized address allocation mechanism. Finally, the alternative solutions for launching an IPv6 reconnaissance attack are briefly discussed.

Chapter 3 answers research question two, with a discussion of the possibilities of launching an address scanning attack in a remote IPv6 network. It is briefly explained why traditional address scanning attacks are less feasible in an IPv6 network and some new IPv6 address scanning techniques that have been proposed from the previous studies are reviewed, such as reducing the address searching space by observing the predictable patterns from the IID field and gathering IPv6 host addresses from the DNS reverse zone. Two large scale surveys for demonstrating the feasibilities of doing the address accountability and DNS reverse searching in the current IPv6 network were carried out. Moreover, some existing reconnaissance tools (Nmap, Metasploit) used in IPv6 networks for launching IPv6 reconnaissance attacks were tested, none of them are able to do "brute-force" address-scanning attacks in an IPv6 network.

Chapter 4 presents an overview of the use NIDS of and HIDS in network security solutions and outlines the similarities and differences between both solutions. Also explained are the signature based and anomaly based IDSs based on the design architectures followed by the pros and cons of two IDS approaches.

Chapter 5 explores the overview of three IDSs (Snort, Bro and Suricata) as well as architectures and algorithms to process the incoming packets. It also demonstrates the similarities and differences of the packet capturing mechanisms that are used in the three IDSs.

Chapter 6 answers research question 3 and 4 and an overview of three popular open-source IDS is provided along with their comparative performance benchmarks. The feasibility of using any of the three IDSs in a high speed network is then evaluated. The limitation of using each of the three IDSs with the default configuration in the high speed network is demonstrated, and recommendations are made of how future IDSs should be designed to be compatible with high speed networks. These include: multi-threading, more efficient packet capture methods and handling encrypted traffic. Moreover, when the existing detection mechanisms in the three IDSs are analysed and evaluated, it is shown that none of them are able to detect the IPv6 DNS reconnaissance attack or new IPv6 attacks. This strongly suggests that IDS developers need to put more emphasis on IPv6 detection mechanisms in future releases.

Chapter 7 presents the results and conclusions of the study. Further research topics are then discussed.

Chapter 2

Literature Review

Network security did not figure prominently during the early stages of network development, but because information technology grows quite quickly, businesses, banks, schools, and individuals have been especially frequent targets of network attacks. People who use the Internet are forced to protect their property and information from continual attacks by Trojan Horses [36], Password Cracking [37], Denial of Service (DoS) [38] attacks and Address Resolution Protocol (ARP) poisoning [39] attacks, etc. When we reviewed current security issues [19, 40, 41], we noticed that some existing attacks leverage deficiencies in current protocols: that is to say, some security concerns were not considered at the time of writing of the protocol RFCs and other security issues were introduced during the period in which the network was developed and deployed. Fine nuances between specifications and particular implementations can be exploited by attackers. In this chapter, we briefly discuss the security issues that still appear in both IPv4 and IPv6 networks with their possible solutions, and then we highlight new security issues in IPv6 networks. Understanding these vulnerabilities will help us in planning and deploying an IPv6 network successfully. The remainder of this chapter is organised as follows: Section 2.1 explains the primary goals of launching attacks based on confidentiality, integrity and availability Section 2.2 discusses the security issues related to both upper layer and network layer protocols, we explain that why those existing attacks still happen in IPv6 networks Section 2.3 demonstrates what new issues are observed in IPv6 networks, such as ND cache exhaustion attacks, or extension header attacks Section 2.4 shows our findings and conclusions

2.1 Common reasons for launching an attack

Today's attacks are launched primarily to obtain secret and restricted information from users system [19, 40]. The disclosure of such information could expose customer privacy information, disrupt business planning, and or threaten national security. By reviewing some recent papers [36, 37, 38], we classified well-known attacks into three categories, based on the purposes of the attack.

- Interruption: most online services need to be available 24/7 for authorized users

to access when they need them. This availability can be affected by attackers who launch attacks to make the system unusable: for example, Denial of Service (DoS) and Distributed Denial of Service (DDoS) attacks are examples of interruption attacks. This type of attack breaks the system availability

- Interception: some data should be hidden from unauthorized users, such as usernames, passwords, or personal information. However, some attackers are able to see and store that private data by monitoring traffic between legitimate users: for example, Man in the Middle (MITM) attack. This type of attack breaks the system confidentiality
- Modification: some threats do not bring a system down, but access the system through system vulnerabilities then modify or delete system resources, such as user information or some important data. This type of attack breaks the system integrity

2.2 Common attacks against the existing networks

If we could have predicted the explosion of the Internet in the early 90s we would have incorporated more security measures into its design. For the original design of the Internet protocols, designers gave only minor consideration to security; in fact, some applications do not follow recommended specifications of how the protocol should be implemented, hence most existing attacks are against protocol vulnerabilities and application weaknesses. In this section, we discuss some common attacks that happen in IPv4, but also appear in IPv6 networks. We start from the application layer vulnerabilities and move to the network layer vulnerabilities.

2.2.1 Application vulnerabilities

Numerous application attacks focus on the violation of confidentiality and integrity, these include web application injection attacks [42], virus and worm propagation attacks [43], and memory overloading attacks [44].

Hypertext Transport Protocol (HTTP) [45] is the core protocol for creating communication channels between users and online service providers. However, some existing configuration settings do not analyse the content of HTTP transmissions. Some previous studies have demonstrated [46, 47, 48] that insecure settings lead to attackers breaking websites by using some well-known strategies: the Cross-Site Scripting (XSS) attack and the Structured Query Language (SQL) injection attack are examples.

The XSS attack [49] allows attackers to bypass the access policy from web sites allowing them to gain permissions for modifying the resources on the system. For instance, attackers can inject an innocent-looking Universal Resource Locator (URL) into a submission request that contains XSS vectors. If a victim clicks the link, it will trigger the browser to execute the embedded malicious script inside that link. Another criterion occurs when a website allows unauthorized users to post a HyperText Markup Language (HTML) formatted message without proper security checking of the message contents. For example, if an attacker uploads malicious codes that appear as 'normal data' to a

server, the server will permanently display this data. If an unsuspecting user visits that webpage, the user's browser will receive the malicious script and execute it automatically. The XSS attack is a classic attack that injects malicious scripts into web servers and launches attacks on the client side.

An SQL injection attack [50] targets an SQL server that allows unauthorized users to inject SQL commands. Normally, attackers find websites that require usernames, passwords and email addresses to launch this attack. For example, if an attacker enters a fictitious email address with a single quotation mark as part of the data and the system responds with an Email address unknown message, it indicates that the SQL server has implemented filtering mechanisms and the attacker can't inject malicious SQL commands. But if the system shows a server failure message, it shows the attacker that an SQL injection attack can be launched in this site. In general, the attacker sends the SQL statements as user input and extracts the user information from the database, or modifies the existing data inside the database.

In last decade, XSS and SQL injection attacks have been widely used and pose a significant risk to web applications, such as Talk Talk [19] incident, where the personal records of over 150,000 customers were compromised in an attack on British telecom company TalkTalk. Many existing solutions, such as [51] and [52] have been proposed to detect and prevent the vulnerabilities that are exploited by the XSS and SQL injection attacks. For instance, [51] proposed a Cross-Site Scripting Secure Web Application Framework (XSS-SAFE), Gupta et al. inspect the features of JavaScript, generate rules and insert sanitization routines for detecting and mitigating XSS attacks. Singh et al. [52] suggest creating a cryptographic hash function for generating a username and password field. The client enters the username and password for the server side verification, only valid users can then access the database.

Virus propagation: Recently, more and more attacks are observed in which a Trojan Horse program is injected into a user's PC. The name 'Trojan Horse' comes from an old Greek story, which tells of how Odysseus and his soldiers hid in a giant wooden horse and conquered Troy when the defenders pushed it inside their own walls. Nowadays, 'Trojan Horse' means that attackers embed some malicious code in some existing application or create a program that functions as a normal application, but contains some abnormal behaviours. When victims download these programs from some unauthorized websites and install one on their systems, the malicious codes run automatically. Again, a Trojan Horse program behaves in the same way as the legitimate program, but it performs some extra functions, such as tracking the user's activities, recording sensitive information, opening a trap door or cracking passwords. We introduce two common Trojan Horse attacks as follows:

Password cracking [37]: refers to some applications that decipher password files. Password cracking programs use well-known dictionaries to generate a guessed password, and compare the result to each record in the password file. When a match is found, the

password is found.

Backdoor attack [53]: describes a Trojan Horse program that opens illegal remote access for an attacker, which will allow the attacker to access the computer bypassing normal authentication processing. Also, the attacker can gain privileged access to view system resources on the targeted computer. These types of attacks are hard to detect, and their influence can be both serious and widespread.

Nowadays, password cracking and backdoor attack incidents are being detected almost daily, such as in 2014, Hugh Son and Michael Riley [40] reported that a J.P. Morgan Chase & Co. server had been cracked through an employee password, hackers exploited and accessed information from 76 million households and 7 million small businesses. One year later, Juniper discovered "unauthorized" code [54] in some firewall products, which would allow attackers to take complete control of Juniper NetScreen firewalls running the affected software. By reviewing the previous studies [55, 56], many solutions have been proposed to detect and prevent these password or backdoor vulnerabilities. Gutierrez et al. [55] used a physically unique function or a hardware security module at the authentication server, the new solution stops the password cracking by checking the physical access to the authentication server. Alminshid et al. [56] proposed a Stepping Stone Detection (SSD) approach to detect backdoor attacks by checking the directionality and inter arrival time of packets.

2.2.2 Network vulnerabilities

An attacker launches network layer attacks to exhaust network resources or break network structures. They believe that destroying one network or server will cause hundreds or thousands of devices to stop working, such as SANS et al. [41] reported that 222,000 users lost power for few hours after network attacks. By reviewing recent network attacks, we classify those attacks into four categories: reconnaissance attacks [28], MITM attacks [57], DoS attacks [58] and spoofing attacks [59].

2.2.2.1 Reconnaissance attack

The first phase of an attack lifecycle involves reconnaissance of a target, which allows attackers to gather information about how many live hosts are in the target network, which ports are open, what operating system vulnerabilities exist, and what types of services and protocols are running on each host. This information can be leveraged by attackers to determine the easiest way to penetrate the defences, and even to assess whether the attacks may succeed. We introduce two well-known host scanning mechanisms that attackers have used to find their targets.

Ping sweep [60] is a method to investigate the range of live IP addresses in a targeted network. Attackers gain the network address of a target network and send Internet Control Message Protocol (ICMP) messages to that network with an increasing sequence of host IP addresses. The purpose of launching this attack is to find live machines in the target network by counting the successful ICMP responses. Such information about live hosts

will be used for future reference. For example, if the host network is 130.216.37.0/24, we can type

for /l %i in(1,1,254) do ping -n 1 -w 100 130.216.37.%i

In a Microsoft command line; this command instructs the computer to ping the IP range from 130.216.37.1 to 130.216.37.254. Figure 2.1 shows that the results after performing a ping sweep command. If the host is not online, the response shows the request timed out, otherwise, it shows the response from the destination node and the Time to Live (TTL).

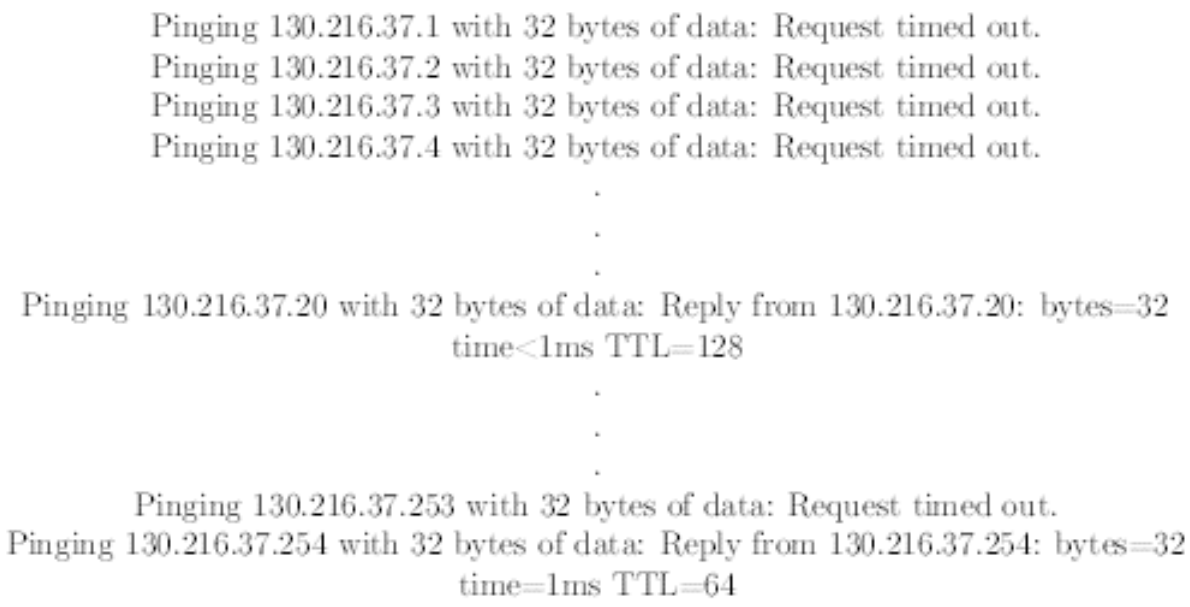

Figure 2.1: Ping Sweep results

A port scan finds open ports on targeted hosts. Such information helps the attacker to know which services are running on the target host. Knowing the open ports, attackers can launch different attacks against some specific vulnerable service. For example, a security report from Veritas [61] discovered vulnerabilities on port 5634, the port is used for communication between the Central Management Server (CMS) and Managed Host (MH). If any host opens this particular port, it is possible for an attacker to access the host through this vulnerability. In March, 2016, North American Electric Reliability Corporation's Electricity Information Sharing Analysis Center and SANS Industrial Control Systems [41] reported that an attacker had launched a series of reconnaissance attacks on the Ukrainian electric utility's system for six months to understand the utility's supervisory control and data acquisition networks, before starting a coordinated series of real attacks on December 23 2015. These attacks caused 225,000 customers to lose power for a few hours. Nmap, ipv6toolkit and thc-ipv6 are some example tools that can be used to launch reconnaissance attacks either in IPv4 or IPv6 networks. Network administrators wish to detect any port scan of a system because that knowledge gives them a hint about attacks that might follow, and some time to prepare to prevent it. Open source IDSs offer

solutions for detecting reconnaissance attacks. For instance, [62] Al-Jarrah et al. introduced two pre-processing plug-in modules (a host sweep pre-processor and a port scan pre-processor) into the existing IDS solution that collects anomalous packets by analysing the temporal attack behaviours from a Time Delay Neural Network (TDNN) structure. Patel et al. [63] proposed a new rule-based IDS solution based on self-generated new Efficient Port Scan Detection Rules (EPSDR). In their solution, a Basic Analysis Security Engine has been introduced into the existing Snort solution for reducing false positive alarm and providing a graphic user interface.

2.2.2.2 Man in the Middle attack (MITM)

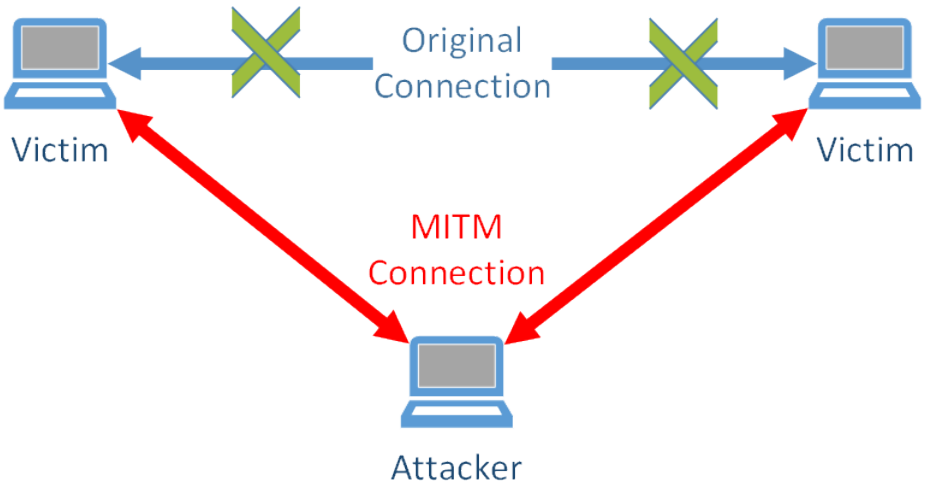

Figure 2.2: Illustration of man-in-the-middle attack

In some cases, attackers will establish independent connections with victims and redirect packets between them (see Figure 2.2). The incoming and outgoing packets could contain credit card numbers, passwords, and personal security information. In this process, an attacker will make the victims believe that they are talking directly to each other over a private and secure connection. Normally, this is called an MITM attack. The MITM attack can succeed if clients initiate a communication without any authentication checking. Previous studies have shown that MITM attacks can take many forms; Below we introduce some common MITM attacks.

A 'replay attack' [64] can be launched by intercepting and storing a legitimate transmission between two hosts. This type of attack normally occurs when an authorized client sends an encrypted login message to a server; that message is captured and saved by an attacker. Even though the message is encrypted, it may still be sufficient for the attacker to gain access to the network by retransmitting the valid logon message. Figure 2.3 demonstrates an example of the replay attack. In addition, the 'replay attack' is a possible accessway for spoofing attacks. If an off-path attacker has observed and captured a Local Area Network (LAN) packet; the attack falsifies the datagrams intended for other hosts.

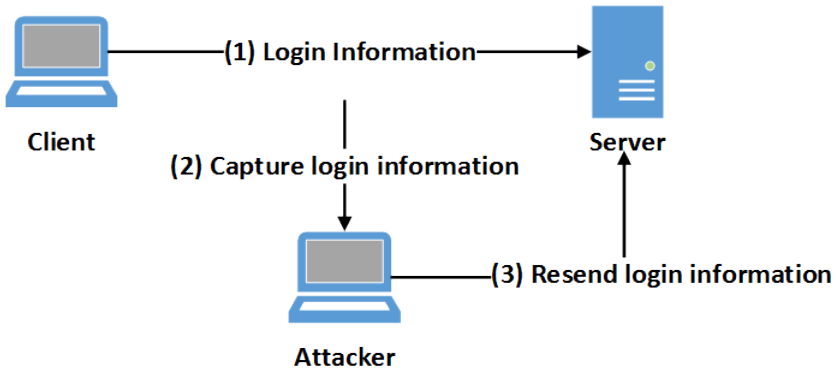

Figure 2.3: Illustration of replay attack

An Address Resolution Protocol (ARP) poisoning attack [39] exploits vulnerabilities in the ARP [65] by sending spoofed ARP messages onto the LAN. The cached ARP addresses are used for resolving the network layer addresses (IP addresses) into Media Access Control (MAC) addresses. In general, the ARP sequence starts from a source node, when the source node checks for a destination's IP address in its cached ARP table; if a MAC address is found, it sends the IP packet on the link layer to a destination node via the local network cabling. If the cache did not return a record for that IP address, it broadcasts an ARP message to discover who owns that IP address. All nodes on the same link are able to receive this request, but only one returns a unicast message to the source node with its MAC address. However, the existing ARP protocol only accepts the first response and ignores the rest. Therefore, an ARP response can come from a malicious node rather than the one with the required MAC address. For instance, Bob meant to send a message to Alice (10.0.0.7/bb-bb-bb-bb-bb-bb); however an attacker has sent a spoofed ARP response to Bob, and Bob therefore thinks that Alice's MAC is [cc-cc-cc-cc-cc-cc]. In this scenario, the attacker intercepts data frames between Bob and Alice and modifies the traffic or stops the traffic altogether. In another scenario, attackers could just send ARP announcements for updating any cached entries in the ARP tables of other hosts that receive the announcement message. For example, Bob has a mapping in his ARP cache for Alice's MAC address and IP address. However, the attacker sends an ARP announcement message to Bob, changing Alice's IP to MAC address mapping to the attacker's IP and MAC address. Again, the attacker can easily redirect the traffic between Bob and Alice.

Many MITM attacks have been detected or reported by previous studies, such as in 2007 [66], when Indiana University detected an ARP spoofing attack at a shared hosting site. In 2013 [67], a Belarusian traffic diversion was reported by Dyn Research, they detected a large amount of traffic being redirected to Belarusian ISP GlobalOneBel, most the traffic was from financial institutions, governments and network service providers. This affected countries across five regions.

By reviewing the previous studies, we find that many solutions have been proposed to reduce or mitigate the MITM attacks. Ali et al. [68] suggested using a source port

sequence as an authentication key for verifying the request from valid clients. In this approach, a server disables the Secure Shell (SSH) service, then sends a packet that includes a pre-defined source port sequence, and the server accesses the source port sequence from the received packet. Only the valid client knows which source port ranges are to be used. For instance, a client sends a message to the server, the message includes a predefined source port sequence 6600, 4500, 2000 and 6779 and the server's destination port. If the message comes with a valid source port sequence, the server will start the SSH service and establish the connection with the client; otherwise, it will drop the connection. A similar solution has been mentioned in [69] McPherson et al. with the introduction of a port-based address binding mechanism that assigns a given port a MAC address. Again, Limmaneewichid et al. [70] proposed a P-ARP solution in 2011, they introduced an authentication scheme for securing ARP. It updates the ARP cache table by adding authentication data for verifying the received ARP packet.

2.2.2.3 Denial of Service (DoS) attack

The motivation for the DoS attack is to make a machine or network resources unavailable to its legitimate users. Malicious attackers can bring a target machine down by sending high volumes of useless traffic. As a result the machine starts to respond quite slowly, and sometimes the machine is unable to accept any incoming or outgoing traffic. The DoS attack generally targets commercial sites or servers, such as banks, credit card payment gateways, or root name servers. However, recent reports show that this technique has been extended; instead of launching the attack from a single host, attackers will compromise multiple systems to flood the network bandwidth and resources on the targeted systems at the same time. That is known as a DDoS attack [71]. Compromising the PCs is the first phase of the DDoS attack, then the attacker controls zombies (the compromised hosts) to generate the volume of traffic for flooding the victim system's resource. In the following paragraphs, we will introduce some well-known DoS attacks.

SYN flood [72] is a transport layer attack; the standard TCP protocol involves 3-way handshake processing between two computers. The TCP handshake mechanism requires two PCs to exchange SYN, SYN-ACK and ACK messages before transmitting data. If the PC at the receive side hasn't received an ACK message from the sender, it will save the current SYN request into a SYN queue and delete it when the SYN timer expires. The SYN flood attack aims to overload the SYN queue by only sending the SYN packets. For instance, an attacker sends a SYN packet to a target host; when the host responds with the SYN acknowledgement, the attacker does not respond with an ACK packet for the current handshake processing; instead it sends a new SYN packet to create a new handshake session. The attacker continues to send new SYN packets until the SYN queue of the target host is filled and the host PC cannot accept any further new connections.

DNS amplification attack: occurs when an attacker spoofs the IP address of the victim host and sends a DNS name lookup requests to an open DNS server. The spoofed packet requests all known information from the DNS server by configuring the query type to

'ANY'. The large volume of response traffic may bring the victim host down by filling the pipe with a large number of DNS responses. This kind of attack takes advantage of the fact that DNS response messages may be substantially larger than DNS query messages and are legitimate data coming from valid DNS servers, therefore it is quite difficult to prevent this attack.

DDoS attacks using amplification/flooding techniques are still popular and allow cybercriminals to break their targets in a short period. For instance, the report from CloudFlare [73] indicated a DNS amplification attack that targeted the Spamhaus website. The attackers used 30,956 open DNS resolvers to generate a 300 Gb/s DDoS for hitting their network. Another DDoS attack has been mentioned by Matthew Dunn in 2016 [74], they observed a TCP SYN flood attack in their data centres. This affected some Zendesk and Zopim customers, who lost network connection. Many security solutions have been proposed to mitigate the damage caused by amplification or flooding attacks. Jose et al. [75] introduced one-to-one mapping to distinguish a normal DNS request from a DNS amplification attack; they embedded a query ID in a DNS request and a DNS response, if the request and response appear for the same query ID, then it is a normal DNS operation, otherwise, it is an attack. Arshadi et al. [76] claim to use the entropy of SYN packet inter-arrival times as a measure of randomness to detect SYN flooding attacks. The method reduces deviations by applying a sliding window on the SYN packet inter-arrival time. Moreover, it introduces a threshold time T and uses T to indicate whether or not traffic behaviour is normal. Other methods use TCP SYN-FIN (RST) or SYN-ACK pairs to detect SYN flooding attacks.

2.2.2.4 Spoofing

In general terms spoofing attacks refer to an attacker falsifying someone's identity to gain access to a system or network. There are many different kinds of spoofing attacks in the existing network, which include IP address spoofing, session hijacking, etc. IP address spoofing: Bi et al. [77] explained an attack against the IP network protocol. The attack will succeed if a destination node does not verify the authenticity of source IP addresses. In the existing TCP/IP network, the IP address is a numerical label assigned to each network device. Some network devices implement an address-filtering mechanism, which will allow registered remote IP addresses to access an internal network. If an attacker sniffs an IP address that is permitted access through the targeted network, the attacker can bypass the address filtering mechanisms by faking a legitimate IP address. The IP address spoofing attack can also evolve into a DoS attack. For example, if an attacker sends a request to a victim by faking the trusted host's IP address. In this case, the response messages will redirect to the trusted host. If the attacker sends a large number of requests to the victim, the response messages may overload the resources on the trusted host. Session hijacking: Wang et al. [78] explained that an attack breaks an existing authorized TCP connection and redirects its legitimate packets to a malicious node. In our current network, the TCP protocol uses a sequence number to determine

lost packets or the order of packets. The initial sequence numbers are generated by the predefined pseudo-random algorithm and increment for each transmission. However, the drawback of having a predictable initial sequence number can be utilized by attackers to launch session hijacking attacks. Figure 2.4 is a visual representation of a session hijacking attack. There are three steps for launching a successful hijacking attack. To begin with, an attacker monitors the exchange of initial sequence numbers on a network and predicts the next set of sequence numbers, then the attacker fakes host A's address and send a reset message to host B, dropping the current connection. After breaking the existing connection, the attacker can start to inject malicious commands into host A by faking host B's IP address.

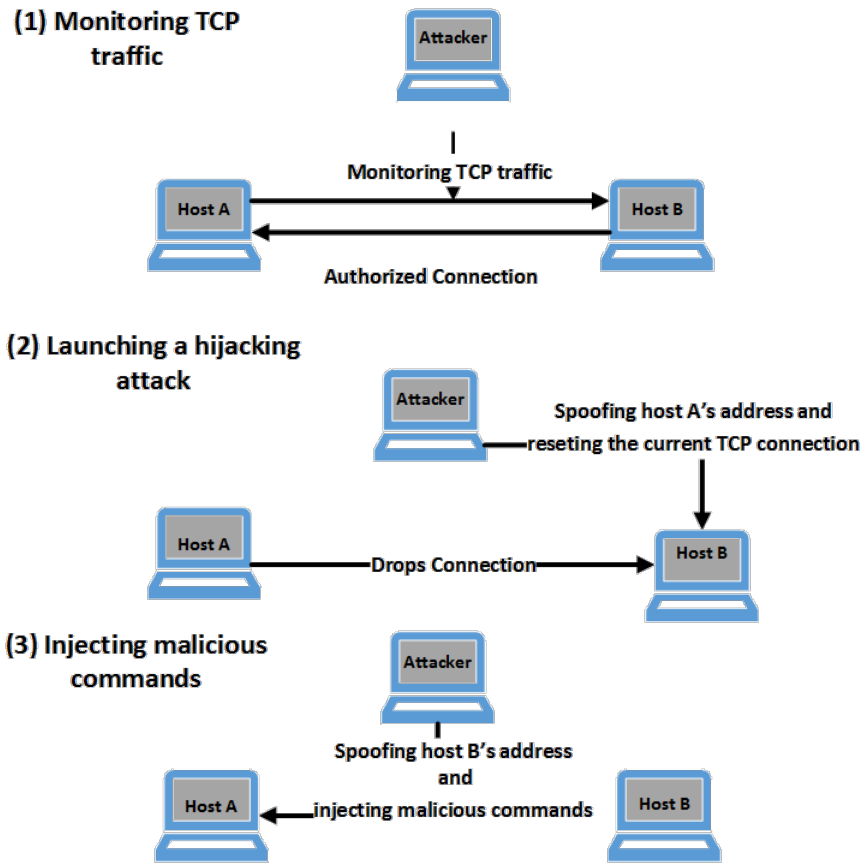

Figure 2.4: Illustration of Session Hijacking

Spoofed IP addresses have been used for launching other attacks, such as on the Network Time Protocol (NTP) [79], Simple Service Discovery Protocol (SSDP) [80] and DNS servers [73]. In 2015 [81], Arbor reported a 334 Gb/s attack against a network operator in Asia through a faked source IP address. A year earlier, Michael Mimoso [82] reported that attackers leveraged the Heartbleed Openssl vulnerability to hijack a client's SSL virtual private network connection. Our study shows that more than one solution is used to mitigate the spoof attack. For instance, Ferguson et al. [83] introduced an ingress filtering mechanism that prohibits an attacker from using forged source addresses by checking the addresses do not exist within the local network segment. In [84] Alabrah

et al. claimed a new approach to reduce session hijacking attacks; they used the client's machine to compute hash values for authentication tokens that are fetched as needed. In their approach, sparse caching techniques are used at the client side; a weighted overhead formula is introduced to obtain insight into the suitable cache size for different classes of mobile devices. They have tested sparse caching schemes with different scenarios, such as testing the sparse caching with uniform spacing, non-uniform spacing and geometric spacing. Their results show that the sparse caching can significantly reduce the overhead of one-way hash chain authentication.

In the next section, we used some existing penetration testing tools (Nmap[1], Metasploit[2], et) in IPv6 networks for simulating existing styles of attack. We observed some attacks have less success in IPv6 networks, because some IPv4 functions are not used in IPv6. For instance, ARP is being phased out in IPv6, therefore ARP attacks do not appear in IPv6 networks. Instead, similar new styles of attack have been detected in IPv6, such as the Neighbor Discovery spoofing attack. In addition, some older tools have less ability to launch attacks in IPv6 networks: Metasploit, for example, does not automatically detect IPv6 addresses during a host discovery scan. To discover individual IPv6 addresses, some new tools have been created for IPv6 address reconnaissance attacks, such as an IPv6Toolkit-v2.0[3] package that provides an option to specify the target address prefix/range of the address scan, and a THC-IPv6 toolkit[4] that supports multiple methods to probe the IPv6 host addresses from remote networks. These tools will later be examined more closely. Furthermore, we find that the security issues related to network hardware and software will still pose a threat to IPv6 protocols. In addition, the attacks against protocols that are above the network layer will still appear in IPv6 networks, because the replacement of IPv4 with IPv6 only changes the network layer.

2.3 IPv6 Vulnerabilities

The IPv4 protocol dates from the 1980s. It has provided a robust, easily implemented and interoperable environment for the Internet's rapid growth since then. However, the 32-bit address space of the IPv4 protocol is quite small. The current rate of Internet growth has exhausted the 4 billion unique IPv4 addresses. In order to solve this problem, in the early 90s IETF designed a new protocol named IPv6. It introduced a larger address space, a 128-bit address space that not only eliminates the shortcoming of IPv4's address system, but also reduces the chance of falling victim to reconnaissance attacks. These types of attacks are carried out by using the ping sweep and port scan mechanisms. The range of IPv4 subnet addresses numbers hundreds or thousands. If we assume an address scanning rate is one million addresses per second, it will take less than one second to find all the addresses in an IPv4 subnet. However, IPv6 has 2^{64} addresses for each subnet. If network

[1]https://nmap.org/
[2]https://www.metasploit.com/
[3]https://www.si6networks.com/tools/ipv6toolkit/
[4]https://github.com/vanhauser-thc/thc-ipv6

administrators use unpredictable values to allocate IPv6 addresses, it will take more than 500,000 years to find all the hosts from one subnet [24]. The larger address space makes a significant contribution to reducing the success of reconnaissance attacks. In addition, T. Aura [85] proposes a Cryptographically Generated Addresses (CGA) mechanism to verify the address ownership. In this solution, an authentication channel is established between senders and receivers, each receiver verifies IPv6 address ownership by checking the IPv6 address and CGA parameters. However, the CGA mechanism is not widely deployed in IPv6 networks, because the implementation quite difficult. By reviewing the existing papers [86, 87, 34] and simulating existing attacks in IPv6 networks, we noticed that some new attacks have been created for IPv6 networks. In the following sections we briefly discuss the security issues that are only detected in IPv6 networks.

2.3.1 Attacks against ICMPv6

In the existing IPv4 network, the ICMP protocol is not used for Neighbor discovery or Router discovery, instead, it provides features for network engineers to test or troubleshoot the IPv4 network, verifies end-to-end IP connectivity and specifies better routing paths out of a network. However, attackers utilize those advantages to launch various types of attack, such as to ping a specified IP range to find live hosts, or forging ICMP redirect packets to launch a MITM attack. In IPv6, the ICMPv6 protocol plays an important role in finding a destination node, or checking the duplicate address. Therefore, it is targeted by attackers for launching MITM or DoS attacks in IPv6 networks. In the following section, we introduce some attacks that leverage the ICMPv6 protocol.

2.3.1.1 Neighbor discovery spoofing attack

Neighbor Solicitation (NS) suffers from the same lack of authentication as ARP on IPv4 networks. In IPv6, if a source node wants to find a destination node in the same network, the source node checks a neighbour cache to find an IP-MAC pairing. If the MAC address is not in the neighbour cache, it sends an NS message asking which host has this IP address; the message uses the IPv6 reserved link local multicast address. All the hosts in the same link will receive the request, but only the host that has this IP address will unicast a Neighbor Advertisement (NA) response. The source node will update its neighbour cache once it receives the NA. However, this is an inherently insecure process, because the IP-MAC pairing information can be accepted without any verification. In this scenario, a malicious node can send forged NS/NA messages to a target node to add a falsified IP-MAC pairing into the neighbour cache at the destination host. Therefore, the spoofed IP-MAC pairing will affect the destination node, so that it will redirect all the data link frames to the malicious node. The figure below illustrates the steps involved in launching a neighbour solicitation attack.

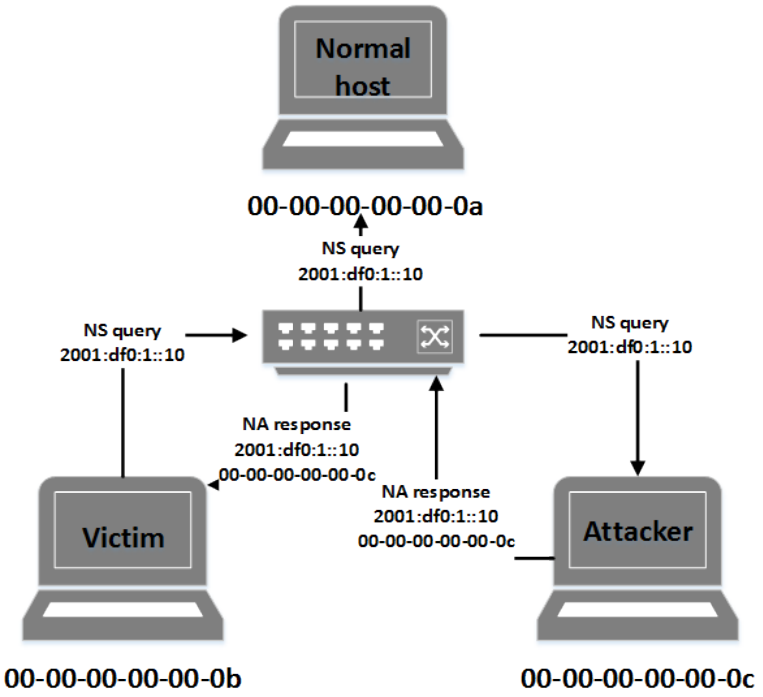

Figure 2.5: Neighbour solicitation attack

Figure 2.5 shows that three hosts on an IPv6 network, a victim (IPv6: 2001:db8::2/64, MAC: 00-00-00-00-00-0b), an attacker (IPv6:2001:db8::3/64, MAC: 00-00-00-00-00-0c) and a normal user (IPv6:2001:db8::1/64, MAC: 00-00-00-00-00-0a). The victim node sends an NS request to query about a MAC address corresponding to 2001:db8::1. In the normal procedure, the user responds with a NA that contains the corresponding MAC address. However, in this scenario the attacker also sends a spoofed NA to the victim having IP (Normal user)-MAC (attacker) mapping, if the response from the attacker arrives first, the victim will update the neighbour cache with a falsified pair IP (Normal user)-MAC (attacker). In other words, the first NA response will update the mapping time, and the rest of the responses are ignored. Now all traffic from the victim to the normal user can go through the attacker.

Some solutions [88] and [89] have been proposed, for instance Nikander et al. [88] suggested the network restricts access to the trusted nodes, or adding a trusted operator in the network, then the operator certifies the address bindings for other local nodes. Again, the router can also play the role as a trust proxy for nodes in the local link. In an ad hoc network, network administrators can use CGA [85] to authorize address bindings. For instance, a receiver obtains the IPv6 source address from the received packet, then checks the collision count in the CGA parameters data structure, if the collision count is not equal to 0, 1 or 2, the verification stops, Again, the subnet prefix in the CGA parameters must be equal to the subnet prefix (the leftmost 64 bits) of the IPv6 address. Comparing the Hash1 [85] with the interface identifier (the rightmost 64 bits) of the address, if the 64-bit values differ, the CGA verification fails. More information of CGA can be found in [85].

2.3.1.2 Duplicate Address Detection (DAD) DoS attack

The IPv6 network introduces a new mechanism for hosts to test the uniqueness of an address. The DAD [90] process requires that a new host sends a neighbour solicitation message to verify whether any hosts on the same link have claimed the address that this new host is going to use. If any hosts respond with a message that states the address is taken, the new host needs to claim another address and send a new NS message to check availability. If no hosts send a message to state that the address is taken, then the new host is able to use that address. However, one basic issue when sending the DAD request is that no verification mechanisms are specified for this process. For instance, an attacker may send a message with a claim that the address is taken; in this case, the attacker prevents a new host from receiving a global IPv6 address. For example, a host A sends a NS message asking who is using 2001:df0:0::a. A malicious host T sends an NA response to state that the address is used, even though the address 2001:df0:0::a is an available address. If host A claims another address 2001:df0:0::c and sends a new DAD request, host T again sends a message to state that the address is used. Essentially, host T claims every address that host A attempts to use is an unavailable IPv6 address. In this scenario, the new host A cannot access other IPv6 networks, because no unique global IPv6 address has been assigned to host A. Again, Nikander et al. [88] give suggestions to mitigate DAD DoS attacks in IPv6, they introduce a trusted node that is able to verify whether the NA response comes from an authorized node or not. Additionally, they propose the trusted operator solution; the operator plays an authorizer role to assign addresses for each new node. For an ad hoc network, no trusted operators can be used, nodes cannot trust each other directly, and the traditional authenticating does not work in this environment. Instead Nikander et al. [88] suggest using a self-authorization solution, for instance, each node can use CGA to ensure that they are talking to the node that actually owns the IP address, and not to a node that is spoofing someone else's address.

2.3.1.3 Router Advertisement (RA) Spoofing

IPv6 allows a host to discover routers on the same link via sending ICMPv6 router solicitation messages. A legitimate router responds with an ICMPv6 RA to indicate itself as an available router. However, some previous studies [91, 92] detected router advertisement spoofing attacks against the IPv6 router discovery processing.

Figure 2.6 demonstrates the steps of launching a router discovery attack. In the target subnet, host A sends an ICMPv6 RA message searching for available routers in its local link. The intruder, shown as host T, attempts to insert itself as a default router in host A's routing table. To do that, Host T sends a spoofed RA message to host A and indicates itself as a default router for host A. After becoming the default router, Host T can view all the traffic from host A. However, Host T cannot see the returning traffic from the Internet, because it is not able to spoof the real default router on the same subnet.

Because of the lack of authorization and vulnerability in the router advertisement, various mechanisms have been implemented to counter this effect. Some common mech-

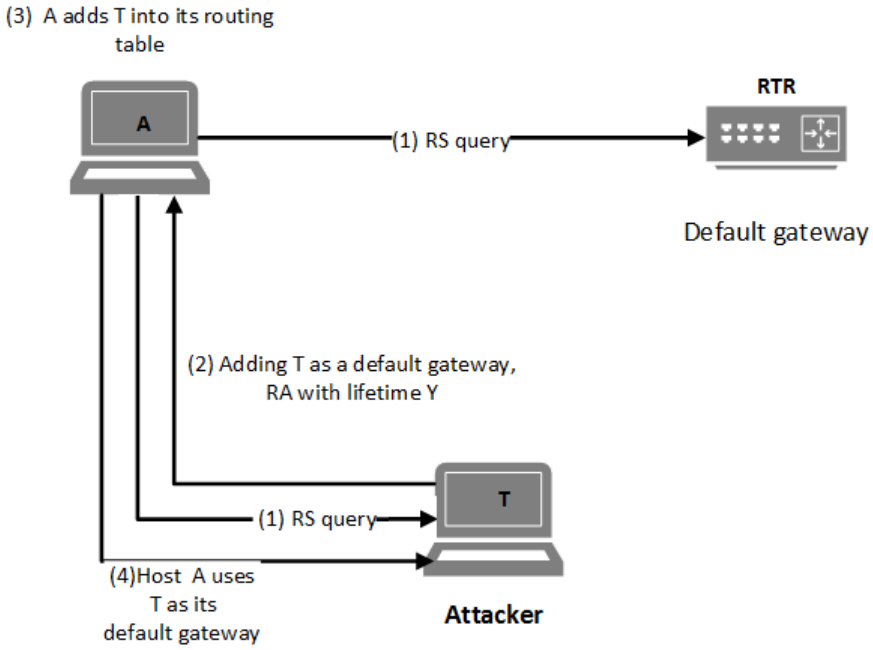

Figure 2.6: A router advertisement spoofing attack

anisms are building an authentication model to show the ownership of the node. For instance Sarma [91] described an Address Based Keys (ABK) solution to identify the ownership of nodes. That solution ensures that the owner of the Network Interface Card (NIC) and its corresponding IP Address has sent the message. This provides message authentication to the message's receiver. Eric et al. [93] proposed the RA guard solution, which can be used in an environment where all messages between IPv6 end devices traverse the controlled L2 networking devices. In their solution, the RA guard filters Router Advertisements based on a set of criteria, such as RA disallowed on a given interface or RA allowed from authorized sources only.

2.3.1.4 Router Advertisement Flooding

In another scenario, attackers can leverage the SLAAC processes on all Microsoft Windows computers to launch a DoS attack. To do this, attackers flood many thousand RA messages and force the operating system to create IPv6 addresses for each RA response. This attack leads the PC CPU usage reaching 100%, so that it needs to power off to be revived. Although this type of attack has been previously detected as a known weakness, it can still happen on Windows machines, such as: NasserElDeen [94] froze a Windows 7 PC by sending a larger number of RAs. The main problem of router advertisement flooding is no trust between the established channels. Attackers can easily send a huge number of RAs. Most of the proposed solutions for detecting the RAs flooding, such as, Najjar et al. [95] propose a machine learning IDS solution; a flooding behaviour is much different from normal behaviour, therefore datasets can be built to store the normal behaviour and compare the incoming traffic with the datasets. Any violation can be treated as an anomaly.

2.3.1.5 ND cache exhaustion

In [29], Gashinsky et al. discussed a potential denial of service condition against the router that performs address resolution on a large number of destination addresses. In IPv6, the subnets are quite big, most existing devices only save assigned addresses. The ND cache table can be prematurely exhausted by sending a large number of ICMPv6 NA responses. Gashinsky mentioned a scenario of launching a DoS attack on a router, where, for instance, a malicious host T starts scanning a ::/64 subnet by sending a packet to the hosts one by one. The router on the same path will attempt to perform address resolution on each received NA. As a result of this, the router exhausts available memory and stops performing the address resolution. In this scenario, the router process of testing for the (non) existence of neighbors induces a DoS condition. Gashinsky [29] has mentioned a few solutions to mitigate the ND cache exhaustion attack. For instance, they suggested filtering unused address space by using Access Control Lists (ACLs) to filter access to addresses not in the ACLs, or they suggested reducing the subnet size.

2.3.2 Attacks against the multicast protocol

In the IPv6 network, it is considerably more difficult than IPv4 to find valid host addresses. However, the 128-bit address space does not mean that the flooding attack is eliminated by IPv6. Some features continue to be a source of problems, such as multicast addresses [96]. Multicast addresses were designed to scale the network transmission by improving point-to-multipoint data transmissions. The multicast protocol allows a source node to send one copy of information to all the multicast nodes. For instance, a source node (A) sends one copy of the information to a multicast destination address. The routers on the path duplicate and forward the information to all members in this multicast group (B,D,E) (Figure 2.7). Finally, Host B, D, E receive the information.

The advantages of using multicast are:

- Over unicast: multicast traffic will not increase the load of the source and will not significantly add to the usage of network resources
- Over broadcast: multicast data is only sent to the users who need it. In addition, the user can send multicast traffic across different subnets; furthermore multicast uses less network bandwidth

2.3.2.1 Using a multicast address to launch reconnaissance attacks

Like the ICMP protocol in the existing IPv4 network, reconnaissance attacks can be launched by sending the request to a multicast group. If members from the multicast group respond to multicast requests, the received responses help an attacker to count how many live hosts there are in this network. Such information can be used for further attacks. In the IPv6 network, Hinden et al. [97] suggested a few reserved multicast addresses, such as the link-local address (FF02::1)for all hosts, the site local address (FF05::2) for all routers and the site local address (FF05::1:3) for all Dynamic Host

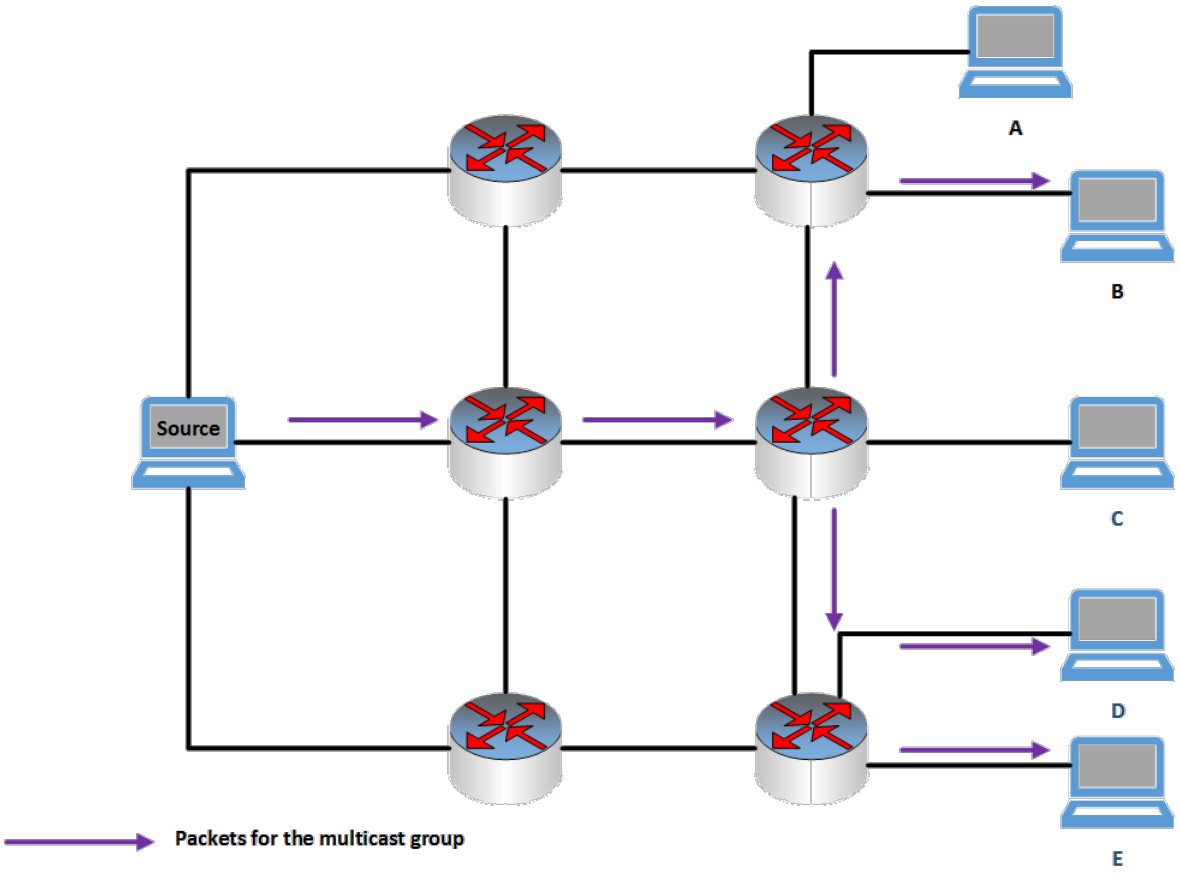

Figure 2.7: A multicast transmission example

Configuration Protocol version 6 (DHCPv6) servers. If an attacker sends traffic to these multicast groups and all the members respond to the request, the attacker would have information about all the routers, local hosts and DHCPv6 hosts within the IPv6 network.

2.3.2.2 Multicast DoS attack

A multicast address can also be used to amplify traffic volumes when launching a DoS attack inside an IPv6 network. For example, an attacker sends a multicast ICMPv6 request that contains a victim's source address; all the members from the multicast group are sent responses to that victim node. Eventually, the large numbers of ICMPv6 responses will slow down the victim PC's processing speed and bring the host down. Naidu et al. [98] address such potential attacks utilizing the IPv6 multicast features and give a firewall solution to help prevent these. They propose to block all variable scopes for multicast addresses. In this way, the firewall could reject all requests that try to ping all the DHC servers that have multicast address FF05::1:3. Again, they also mention that the IDS solution can be used to detect and reduce reconnaissance and DoS attacks in IPv6.

2.3.3 Attacks against the extension Header

Extension headers are a new feature in IPv6 that add separate sub-headers between the IPv6 header and the transport-layer header inside IPv6 packets. All the optional

extension headers are chained together, and each header includes a distinct value for the next header. The advantage of using extension headers is adding some specific matter that cannot be described in the standard IPv6 header. An IPv6 packet may carry zero, one, or more extension headers. The disadvantage of using one or more extension headers is that the transport layer header is forced to shift from its usual packet location to a new position after the extension headers that may cause problems for some existing security solutions, for example, if a firewall is configured to only forward the packet that is transferred to a particular TCP port on a particular IPv6 address, an attacker spans the upper layer header chain into multiple fragments. In this case, the firewall may fail to process its forwarding policy, because the first fragment does not contain enough information for the firewall to enforce its forwarding policy. In order to solve this problem, in [99], Bonica et al. suggest that for each IPv6 datagram, the first fragment packet contains the entire IPv6 header chain that includes the IPv6 header, Extension Headers, and Upper-Layer Header. In another scenario, if network devices are configured to parse all extension header fields until they reach the upper layer header, this may cause system resource exhaustion issues if the network devices have to process a large number of extension headers. For example, an attacker could forge an IPv6 packet with a large number of extension headers, which would cause routers and a destination node along the transmission path to take more time to process the legitimate packet. In addition, if the destination host does not implement correct logic for checking the header, it could potentially cause a software failure when processing the lengthy extension headers.

2.3.3.1 Atomic fragment DoS

In [34], Gont explained a DoS attack that leveraged a fragment header vulnerability. He discussed atomic fragment packets, where the IPv6 packet contains a fragment header without being actually fragmented into multiple pieces. When a host receives a 'Packet Too Big' message, it generates an atomic fragment, each packet contains IPv6 Source Address, IPv6 Destination Address, Fragment Identification. However, this process has been leveraged to perform a DoS attack. For instance, in [100], Krishnan mentioned that IPv6 nodes will discard the fragment packets if the atomic fragment comes with a set IPv6 Source Address, IPv6 Destination Address, Fragment Identification that is already queued for reassembly. So, an attacker forges ICMPv6 'Packet Too Big' error messages, which forces an original host into generating multiple atomic fragments. And then the all the fragments will be dropped by IPv6 nodes because of overlapping fragments.

2.3.3.2 Bypass a firewall with fragment header

Another attack scenario is an attacker reassembling the first packet that has many extension headers and spreading the payload into following fragmented packets, with the result that if the initial fragment passes a firewall, the firewall will not look at the second, third, and following packets. This method allows attackers to send many small fragments to bypass filtering or detection rules. Moreover, attackers can exhaust the resource on the

destination node by sending an incomplete set of fragments so as to force the destination node to spend system resources waiting for the final fragment in the set. For reducing attacks against the extension header, Bonica et al. [99] provided few countermeasures. For instance, they suggested to black all unused extension headers, or use IDS to implement rate limits for hop-by-hop options headers. Again they claimed that a firewall should reassemble the fragmented packets in order to investigate the upper layer information.

2.3.4 Attacks against the co-existence/transition mechanisms

Although IPv6 was slowly deployed in the last decade, and most operating systems can support the IPv6 protocol, many major services still use the IPv4 network. In order to exchange traffic between the two separate protocols, network engineers designed some transition technologies. In reviewing the existing literature, we noticed that the existing transition mechanisms are based on two solutions. In the following sections we briefly discuss how they work and what security issues are addressed.

2.3.4.1 Translation

Due to the significant differences in the protocol format, IPv4 and IPv6 networks are not inter-operable. The IPv4-IPv6 translation solutions attempt to achieve direct communication between IPv4 and IPv6 only nodes. The mechanism uses translating nodes to translate the headers of IPv4 packets into IPv6 and the IPv6 headers into IPv4. The most well-known translation mechanisms can be divided into two categories: stateless translation and stateful translation. IVI is a stateless translation mechanism, an IVI IPv6 address contains a Network Specific Prefix, IPv4 address + suffix. In [101], Bao et al. claimed that a local DNS64 server is used to store the address mappings for both IPv6 and IPv4 hosts. The DNS64 turns the A records of IPv4 hosts into AAAA records based on the address mapping rule. In addition, IPv6 hosts store their IPv4 addresses in the DNS server as A records, this helps the DNS server to process requests from the IPv4 networks. The IVI solution improves a routing scalability problem and addressing issues that happen in Stateless IP/ICMP Translation (SIIT) [102] algorithm. Unlike stateless translation mechanisms that assign IPv4 addresses to IPv6 hosts, stateful translation mechanisms introduce an IPv4 address pool and maintain it on the translator. The most well-known stateful translation mechanism is NAT64 that specifies the communication initiated from the IPv6 side. There are two important components to be discussed in this solution.

- The NAT64 server creates a NAT mapping between the IPv6 and the IPv4 address. The NAT64 server has two network interfaces, one of these interfaces is connected to an IPv4 network, and another is connected to an IPv6 network
- DNS64 inherits functions from the existing DNS structure that resolves a domain name to its corresponding IP address. It translates the AAAA query from IPv6 hosts into A query, and then translates the A response for IPv6 hosts into AAAA

response following the IPv4-mapped address. Figure 2.8 demonstrates an example of using the NAT64 solution

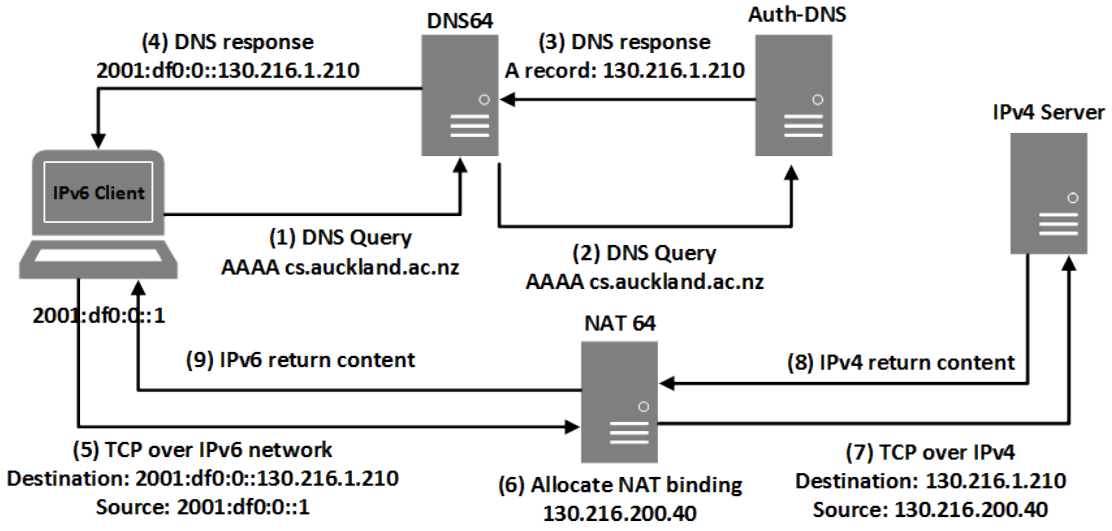

Figure 2.8: An example of NAT 64 and DNS 64 usage

The main concern of using a stateless solution is IPv4 address consumption. To solve this problem, Bao et al. [89] provided an address multiplexing solution that allows one IPv4 address to be shared by multiple IPv6 hosts through port space division. Again the big security issues for stateful translation mechanisms is a DoS attack For instance, if an attacker sends many packets with different faked source addresses, it will exhaust the available IPv4 addresses or ports. The use of an ingress filtering mechanism on the IPv6 side can reduce the problem; the network administrator can simply manually configure the access control lists. Table 2.1 compares the weakness between IVI and NAT64.

Table 2.1: Comparison between IVI and NAT64, showing weaknesses

Mechanism name	**Mechanims status**	**Type of Address mapping**	**IPv6 prefix usage**	**IPv4 address usage**	**Weakness**
IVI	Replace SIIT	Stateless	Same network specific network for IPv6 hosts and IPv4 hosts	One IPv4 address for each IPv6 host	IPv4 address consumption issue
NAT64	Replace NAT-PT on IPv6→IPv4 direction	Stateful	Fixed IPv6 prefix for IPv4 hosts	Translator maintains the address pool	DoS attack risk

2.3.4.2 Dual-stack

The dual stack approach means the host supporting both IPv4 and IPv6 protocols. In order to achieve this, the solution requires hosts or routers to implement both protocols. A dual stack host gets an IPv4 address from the Dynamic Host Configuration Protocol (DHCP), while a IPv6 address can be generated through the stateless address auto configuration. Using a dual stack solution leads to some unnecessary configuration and potential security risks. For example, a router has to maintain routing tables for both protocols, the same process applies for Firewalls. In addition, there are some issues related to having dual stack IPv6 on by default, such as those discussed by Roy et al. [45]. It was shown that some operating systems enable IPv6 by default; if such systems are installed and placed in an IPv4-only network, problems may occur with the neighbour discovery processing. For instance, if a network that has no IPv6 configured router or no IPv6 neighbours, if a host has been enabled the IPv6 by default, it will check the default router list, if the list is empty, the host will assume that the destination is reachable, it will attempt neighbour discovery link-layer address resolution for each destination. This process leads to potential transport layer costs plus connection timeouts. Regardless, the user actually expects that the application can quickly connect to the destination without a waiting period. Furthermore, the dual stack solution opens the opportunity for attacking both protocols, such as the ARP attack in IPv4 and the ND spoofing attack in IPv6.

2.4 Conclusion

As information technologies grow quite quickly, Internet protocols have become the most pervasive and widely used communication protocols in existence. Finding out how to build a secure and stable network has become a challenging task for network engineers and administrators. They must be sure that all packets are safely and efficiently delivered without endangering system safety. Drawing upon the material presented in this chapter, we find that IPv6 is similar to IPv4 with larger addresses. From the security point of view, there is no significant difference between IPv4 and IPv6. In some scenarios, IPv6 is slightly more secure, technologies such as CGA can be used for reducing address spoofing attacks, the privacy IPv6 address allocation mechanism makes the address reconnaissance attack less feasible. However, in other cases IPv6 is less secure, for instance the extension header attack, the ND cache exhaustion attack. Moreover, it is still possible for some existing IPv4 attacks to be launched in an IPv6 network. Table 2.2 summarises the existing IPv4 attacks with the related incidents and the feasibility of launching these attacks in IPv6 networks.

By reviewing previous studies, some new attacks were found against the new methods in IPv6. For instance, a method similar to the neighbour discovery spoof attack was used to launch a MITM attack in IPv6. Again the DAD approach has been used for launching a DoS attack in IPv6. Moreover, a few new threats attempt to attack the first hop security, such as the ND cache exhaustion, the router advertisement spoofing and

Table 2.2: Demonstrating security changes from IPv4 to IPv6 with the relevant incidents

Type of Attacks	**Incident**	**Reconnaissance**	**MITM**	**DoS**	**Spoofing**	**IPv6 feasibility**
Ping Sweep	In 2016, A reconnaissance attack on Ukrainian electric utility's system[41]	Yes				Less feasible
Port Scan		Yes				Yes, if can find a live host
Replay			Yes			Yes
ARP	In 2007, Indiana University detected a ARP spoofing attack at a shared hosting site [66]		Yes			No, but the ND attacks use the similar technology
SYN flooding				Yes		Yes
DNS amplification	In 2013, a DNS amplification attack that targeted Spamhaus website [73]			Yes		Yes
IP address spoofing					Yes	Yes, if the IPv6 address is not generated by CGA
Session hijacking	In 2014, Heartbleed Openssl vulnerability to hijack the client's SSL virtual private network connection [82]				Yes	Yes

the router advertisement flooding. We have tested existing tools to simulate new IPv6 attacks in our experimental environment; most such tools are not able to launch new IPv6 attacks, and instead some new tools have been created, such as:

- THC-IPv6: The toolkit provides many programs to access the IPv6 vulnerabilities, such as fakerouter6 which can fake the RA response and announce itself as a router on a network. Again the floodroute6 will send random RAs to flood the target device. Parasite6 can launch a MITM by sending ICMPv6 neighbor solitication/advertisements
- SI6 Networks' IPv6 Toolkit: This package aims to help IPv6 vendors improve the security and resiliency of their products, for instance: frag6 provides accessing IPv6 fragmentation and reassembly function, it can be used for either flooding a target or accessing the fragment reassembly policy of a given node

In addition, we briefly covered some attacks that are possible against the transition methods from IPv4 to IPv6. The existing transition solutions come with some weaknesses, such as the translation solutions that have scalability issues, the stateless translation solutions that requires one IPv4 address for one IPv6 host. While stateful translation requires a better state maintenance algorithm for the address binding. Neither solution is suitable for larger scale networks. Moreover, the Dual Stack solution increases the operation cost, such as the router needing more memory for maintaining both protocols. Again, the network security requirements became more difficult for protecting both protocols. To sum up from the previous studies, the first phase of launching a successful attack is a reconnaissance attack. In this type of attack, a malicious user attempts to collect as much vulnerability information about a victim network as is needed for launching successful attacks. For instance, running OS information can be leveraged to inject a Trojan Horse program, or the identification of a host with a low memory resource or low CPU speed can be used for launching the DoS attacks. Again, most attackers will compromise a victim host from the target network and use this host to launch the serious attacks.

As we mentioned earlier, the main difference between IPv4 and IPv6 is the address space, IPv6 provides the huge address space that may stop attackers from finding a target host using only the traditional reconnaissance technologies. For instance, if a network administrator uses the privacy address allocation mechanism to assign the IPv6 host address, it will take 500,000 years to complete an address scan in a /64 subnet. However many studies [26, 27] have observed that there are numerous ways for attackers to find targets from IPv6 remote networks. Such as some address allocation mechanisms generate common patterns in its IID field that can be leveraged by attackers to cut the address search space or the DNS reverse zone can be leveraged to get IPv6 host information. In the next chapter, we will introduce a few new IPv6 reconnaissance threats in current IPv6 networks. In addition, we will demonstrate the changes of IPv6 address allocation mechanism usages in last five years and do two large scale network surveys to investigate the feasibility of launching successful IPv6 reconnaissance attacks.

Chapter 3

IPv6 Vulnerability

In the previous chapter, we compared existing attacks and new attacks; we discovered that most attacks share some common strategies, in particular a reconnaissance attack is an initial step toward finding hosts or system vulnerabilities. Our existing IPv4 network is susceptible to two types of address space scanning strategy. One strategy generates a random number to probe new target addresses. Another chooses a target network and sequentially increases the host ID to find active addresses in that network. IPv6 was proposed as a solution in 1996; it extended the Internet address space to 128 bits. The larger address space not only supports an increased number of connected devices, but also makes some reconnaissance strategies less successful in IPv6 networks. However, more and more studies [27, 28, 24, 26] have found that it is possible to find live hosts in IPv6 networks from internal or external networks, for instance: the common patterns in its IID field can cut the search space considerably, a multicast address can be used to discover live hosts in the same subnet, or the DNS reverse zone can be leveraged to get IPv6 host information. In this chapter, we focus on detecting reconnaissance attacks in IPv6 remote networks; we launched passive and active large scale surveys of real IPv6 networks to demonstrate IPv6 Host address allocation mechanism usage trends during past five years. How feasible it is to launch a DNS reconnaissance attack in real IPv6 networks? What are the security concerns of deploying the IPv6 DNS reverse zones? The remainder of this chapter is organised as follows:

Section 3.1 reviews and explains some new address scanning strategies in IPv6 networks

Section 3.2 describes our IID allocation mechanisms survey, we used the passive measurement method to capture incoming and outgoing IPv6 traffic from a link connecting a University of Auckland IPv6 network for three months. Our results demonstrated the IPv6 address allocation changes in past five years

Section 3.3 discusses how to launch an IPv6 DNS reconnaissance survey in a real IPv6 networks We used an active measurement strategy to search the public DNS servers cross five regions and fifty countries for two months. The results shown that DNS reconnaissance attacks can be launched in real IPv6 networks

Section 3.4 demonstrates our findings and conclusion.

3.1 IPv6 address scanning strategies

A larger address space in IPv6 makes traditional address scanning attacks less feasible. In [28], Chown mentions that if an attacker launches a probe per second, it will take 500,000 years to complete an address scan in a /64 subnet. However, more and more studies [27, 28, 24, 26] have discovered alternative ways that attackers can launch IPv6 reconnaissance attacks, such as If attackers and victims are located in the same subnet, an attacker can find the active IPv6 addresses though the neighbour discovery protocol [27] or DNS Zone Transfers. However, if attackers are in another IPv6 network, then they have to consider a few problems, for instance, where to find the live IPv6 host address information, how to enumerate active addresses in the target network and how to reduce the address search space. We will answer these questions in the following subsections.

3.1.1 Searching addresses in transition solutions

IPv6 to IPv4 transition solutions give us more flexibility to pass packets between IPv6 and IPv4 networks. Some special patterns have been added in the IPv6 IID field that can be used to indicate which transition mechanisms have been used, for instance if the host is using 6to4 to allocate an IPv6 address in the Microsoft environment, then the IPv6 address will look like 2002:V4ADDR::VADDR. The network prefix for 6to4 is 2002::/16. Teredo generates the IID field by using the IPv4 address of a Teredo server and a UDP port number. The network prefix for Teredo is 2001:0000::/32. Intra-Site Automatic Tunnel Addressing Protocol (ISATAP) sets the first 32 bits of the IID field to be 0000:5efe or 0200:5efe. However, In [24], Chown pointed out that some patterns have vulnerabilities that can be leveraged to reduce the IID search space. For example, as discussed above, if 6to4 is used to allocate IPv6 addresses in the Microsoft environment, an attacker can reduce the 128-bit IPv6 address search space to a 32-bit IPv4 address space. Moreover, Chown points out that if the host enables dual-stacking, attackers can find a host easily through IPv4 connectivity instead of searching for its IPv6 address. Again, Chown [24] explains the weakness of using Teredo, "the 64-bit node identifier is generated from the IPv4 address observed at a Teredo server along with a User Datagram Protocol (UDP) port number. The Teredo specification also allows for discovery of other Teredo clients on the same IPv4 subnet via a well-known IPv4 multicast address."

3.1.2 Reducing the IID search space

Unlike the address scanning attacks in IPv4, it is impossible to scan the entire /128 bits, but it is possible to reduce the search space, for instance: the IPv6 routing prefix (48 bits) can easily be found from a Regional Internet Registry (RIR) website or from some IPv6 research websites, e.g. IPv6 Deployment Status [103], many other prefixes can also be found in the BGP tables. Additionally, if the network administrators set all bytes in the subnet ID field to be 0, attackers then can further reduce the search space to the

64 bits of the IID. In [97], Hinden et al. specify that an IPv6 unicast address is built by n bits of network prefix and (128-n) bits of IID. The network prefix is used to route packets to a subnet and the IID refers to the ownership of a given interface that connects to the subnet. Because the IPv6 routing prefix can easily be found from the RIR or other organisations, how to fill the IPv6 IID field becomes a key factor to mitigate IPv6 address scanning attacks. By reviewing the existing IID allocation mechanisms, we noticed a few explanations for setting the IID's length to 64 bits long; for instance, Hinden et al. specify the IID length is 64 bits for deriving the IID values from IEEE EUI-64 hardware addresses. Carpenter et al. [104] point out two reasons for setting IID at /64. "One was the original '8+8' proposal [105] that eventually led to the Identifier Locator Network Protocol (ILNP) [106], which required a fixed point for the split between local and wide-area parts of the address. The other was the expectation that 64-bit Extended Unique Identifier (EUI-64) MAC addresses would become widespread in place of 48-bit addresses, coupled with the plan at that time that auto-configured addresses would normally be based on interface identifiers derived from MAC addresses."[104]

Clearly, a limitation of IPv6 address scanning is that it is infeasible to search a randomized /64 IPv6 IID field by using existing tools. However, as Chown et al. discuss in [28], if an attacker finds some common patterns from the IID allocation mechanism, the attacker can then easily reduce the search space in the IID field. For instance, if a network administrator uses an IEEE EUI-64 mechanism [97] to assign the IID field, an attacker can effectively reduce the search space from 2^{64} to 2^{48}, because the EUI-64 contains the common pattern 0xFFFE. In addition, if the attacker knows the organisationally Unique Identifier (OUI) of the network interface card, the search space can be further reduced from 48 bits to 24 bits. So if an attacker probes every second, it takes only 194 days to complete the search in a /24 network. Chown et al. [28] describe several methods to combat this (using common patterns to allocate the IPv6 IID field), such as not using sequential addresses, not using simple patterns, and using random numbers to allocate the IID field.

If network administrators applied these suggestions, that would make the classic network address scans less feasible. But, some previous studies [28, 107, 108] show that network administrators do not use the methods recommended in [28] to allocate their Interface ID fields, instead they use customized IID allocation mechanisms to satisfy different requirements, such as the small integer scheme, the embedded IPv4 scheme, etc (more detail in Section 3.2.1). It is therefore possible to launch a network reconnaissance attack in IPv6 networks by using appropriate heuristics. In order to search for a better understanding of the existing IID allocation mechanisms and demonstrate the IID allocation changes over last five years, we launched a survey of the IPv6 IID allocation mechanism usages in from the University of Auckland campus network. More detail of our survey can be found in section 3.2.

3.1.3 Leveraging DNS Reverse Zone

Because traditional address probing had less success in IPv6 networks, attackers started to search for new ways to gain host address information. Chown et al. [28] believes that DNS servers will become a new target for attackers wishing to explore IPv6 addresses. In principle, DNS is designed to convert a domain name into an IP address; it builds a tree structure to map IP addresses and domain names.

DNS has not been targeted as a resource for exploring addresses in the IPv4 network because it is possible to search the entire 32-bit address space in a few hours. But it is infeasible to do such a brute -force scan in IPv6. Van Dijk [26] discovered that DNS reverse mappings can be used for discovering IPv6 nodes. A reverse lookup attempts to map an IPv4 or IPv6 address to a corresponding domain name record; in [109] Thomson et al. specified that "An IPv6 address is represented as a name in the ip6.arpa domain by a sequence of nibbles separated by dots with the suffix 'ip6.arpa'. The sequence of nibbles is encoded in reverse order, i.e., the low-order nibble is encoded first, followed by the next-lowest-order nibble and so on. Each nibble is represented by a hexadecimal digit. For example, the reverse lookup domain name corresponding to the address 4321:0:1:2:3:4:567:89ab would be b.a.9.8.7.6.5.0.4.0.0.0.3.0.0.0.2.0.0.0.1.0.0.0.0.0.0.0.1.2.3.4.ip6.arpa." [109]

Basically, Van Dijk [26] disclosed that the attacker only needs to walk through the target 'ip6.arpa' zone by issuing queries for Pointer Record (PTR) records corresponding to the domain name, he discusses a searching mechanism to gain IPv6 addresses from the 'ip6.arpa' zone. First, users need to give a network prefix of the domain from the reverse DNS zone that they want to search. When the program receives the network prefix, it adds a new nibble (all the new nibbles start with zero) and appends it to the given domain name, the program then sends a reverse lookup with this new address block to DNS servers. In principle, there are three possible responses from DNS servers:

- 'NXERROR' (RCODE 0) means this '*.ip6.arpa' domain exists in the ip6.arpa domain, but there are no PTR records for it. When the program receives this message, it adds a new nibble and appends it to the previous reverse query. The initial value of the new nibble is 0
- 'NXDOMAIN' (RCODE 4) means there are no records for '*.ip6.arpa' in the domain name space. The program will increase its value for the current nibble and send the request again
- If the response is the hostname, the program will save that hostname into the database

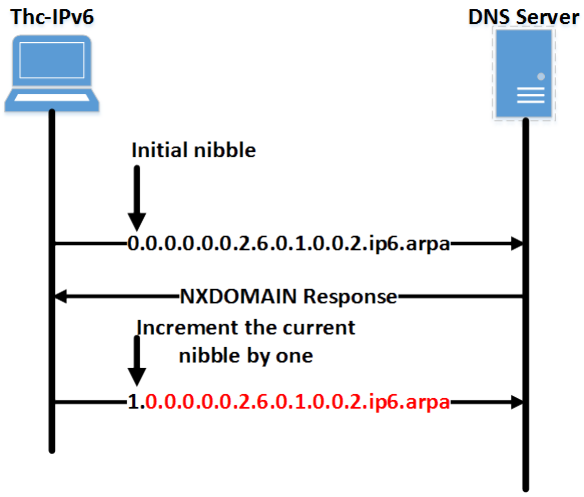

Figure 3.1: The sequence of process 'NXDOMAIN' responses

For example, if an input address prefix is 2001:620:0::/48, Figure 3.1 shows the sequence of process 'NXDOMAIN' responses. If the program receives 'NXDOMAIN' responses, the program will check the current nibble value. If the current nibble is equal to value F, the program will be terminated; otherwise, it increases the current value by one. In contrast, if the 'NXERROR' response comes back, it will create a new nibble to add to the existing IPv6 address and this new nibble starts with zero.

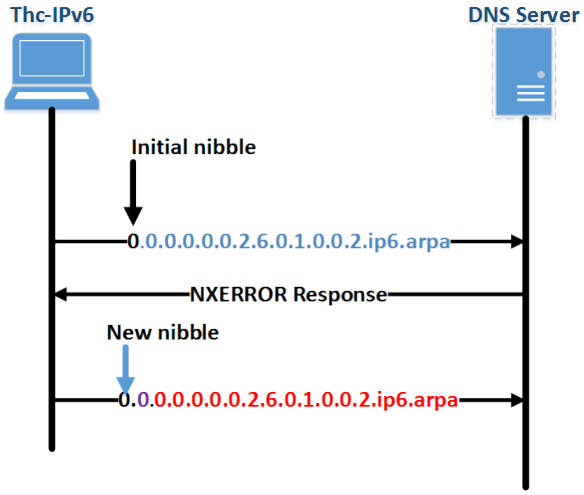

Figure 3.2: The sequence of process 'NXERROR' responses

Figure 3.2 shows the process's sequence of 'NXERROR' responses. In order to search for a better understanding of the existing IPv6 DNS deployment strategies and evaluate the possibility of launching a DNS reconnaissance attack in our existing IPv6 network, we launched a DNS reconnaissance attack survey crossing five regions and fifty countries. More detail of the DNS reconnaissance attack survey can be found in section 3.3.

3.2 How Interface ID allocation mechanisms are performed in IPv6

This section discusses our survey of how IID allocation strategies apply to real IPv6 networks. We will demonstrate the trend of IID allocation mechanism usages in the past five years. Before we consider our survey strategies and results in detail, we would like to discuss some background information. Section 3.2.1 provides an essential description of address configuration mechanisms in IPv6 followed by some common IID allocation mechanism descriptions. We also demonstrate the predictable patterns in IPv6 IID allocation mechanisms that can be leveraged to reduce the IPv6 address search space when performing IPv6 address scanning attacks. Section 3.2.2 explains our survey strategies and compares results of how IID allocation mechanisms have worked over the past five years.

3.2.1 Address Configuration in IPv6

Comparing with the existing address configuration mechanisms in IPv4 (DHCP or manual configuration), IPv6 introduces a new automatic address-configuration mechanism: Stateless Address Autoconfiguration (SLAAC) [90]. The following subsections describe each of the possible configuration mechanisms and IID allocation strategies in detail.

3.2.1.1 Stateless Address Autoconfiguration

SLAAC allows every new host to obtain an IPv6 address automatically when that host appears on the local network. The node uses the pre-configured IID allocation mechanism to generate a link-local address and sends a neighbour a solicitation message to verify whether this is a unique address or not, this process is called DAD. If DAD ascertains that the address is a unique address, then the node applies this address to the interface. The new node then sends router solicitation messages to the routers to request network prefix information. Local routers provide the network prefix, the new host then appends the locally-generated IID to the corresponding IPv6 prefix. Some SLAAC IID allocation mechanisms are described in the following passages:

Embedding IEEE identifier (EUI-64)

In [97], Hinden et al. describe an auto-configuration approach called IEEE EUI-64. This is a traditional SLAAC IID allocation mechanism that uses the link layer address from the corresponding network interface card. Their approach eliminates the need for manual configuration or for DHCP6 to assign the IPv6 addresses for a new host. The mechanism uses three fields. The first 24 bits of the IID field will reuse a network device's OUI bits, followed by the 16 bits 0xFFFE. The remaining 24 bits of the MAC address will fill the lower 24 bits in the IID field. Additionally, the Universal bit of the address is set to 1. For instance, if a host has the MAC address 00:b3:40:90:45:3c and receives a router

advertisement message from the 2001:df0:0::/48 network, after putting the IID value to the corresponding IPv6 prefix, the host gets the IPv6 address 2001:df0::02b3:40ff:fe90:45:3c.

The main advantage of using this solution is that it needs no pre-configuration. Any node can get an IPv6 address when it is connected to the network. Although EUI-64 is easy to use, there are some disadvantages to it, such as reduction of the search space, because EUI-64 introduces two common patterns that cut down the search space in the IID field. In addition, the first three MAC address bytes refer to the OUI; if attackers know the interface card information, they can use the OUI bytes to further reduce the IID search space. We can presume one scenario: if an organisation ordered all its NIC cards from the same vendor and put them into the same subnet, in such a case, if attackers observe a few host addresses in that subnet, they can easily reduce the IID search space from 64 bits to 24 bits and then find all the active hosts. Again, the same scenarios can happen in a virtualization environment. For example, VirtualBox can automatically generate MAC addresses for a new virtual machine; VirtualBox has a static OUI, 08:00:27, for all virtual machines. As a result, if attackers recognize that the host is configured through VirtualBox they need to search only the last 24 bits from the IID field. Moreover, if a VMWare ESX server uses EUI-64 to configure the IID field, the attacker can reduce the search space from 64 bits to 8 bits, because the VMWare ESX server configures the first three bytes of the MAC address to 00:05:59 and the following two bytes reuse the last two bytes from the host's original IPv4 address. The last 8 bits of the MAC address are generated from the name of the virtual machine's configuration file via the hash mechanism.

Temporary and Privacy addresses

As we mentioned earlier, the IEEE EUI-64 mechanism introduces a few security and privacy issues. For instance, attackers can easily predict the address that is derived from the IEEE EUI-64 mechanism. Attackers can also track user activities by monitoring IPv6 addresses, because the IPv6 address structure is divided between a topological portion and an interface identifier portion; if the interface identifier remains the same when a host moves to different networks, it is possible for attackers to track the movements of that host. In order to address this problem, many solutions have been proposed to protect address privacy. For instance, Narten and his colleagues [110] propose a temporary addresses solution for SLAAC in IPv6; they suggest generating a random IPv6 address by putting random IID values and the auto-configuration IPv6 prefix together. The values in the IID field have an expiry time; this solution reduces the exposure of user activity to any third parties. But, Narten et al. [110] also mention that such tactics "do not result in any changes to the basic behaviour of addresses generated via stateless address autoconfiguration. Create additional addresses based on a random interface identifier for the purpose of initiating outgoing sessions" [110]. This means that, assuming an attacker and a victim are allocated to the same subnet, the attacker can still predict the traditional

SLAAC address that is derived from the IEEE EUI-64 mechanism. In addition, regenerating the addresses over time raises network management issues, because the network administrator has to reconfigure the DNS mapping for the static IPv6 servers. In order to mitigate the network management issues, Microsoft extends the aforementioned algorithm and generates randomized stable IIDs to minimise the possibility of an attacker leveraging the predictability of traditional SLAAC addresses. The drawback to this solution is that the host can easily be tracked across visited networks because the values in the IID field remain the same. In order to address these predictability and privacy issues, Fernando Gont proposed a new solution [111] to generate randomized stable interface identifiers in each subnet. When the host moves to other subnets, the interface identifier values must be re-generated, such that the new solution eliminates address predictability issues without sacrificing privacy protection.

3.2.1.2 Dynamic Host Configuration Protocol Version 6 (DHCPv6)

Another automatic address-configuration mechanism in IPv6 is called DHCPv6, There are two configuration mechanisms that can be used in DHCPv6: stateless [112] or stateful. If an IPv6 node derives IPv6 addresses through some other allocation mechanisms (such as stateless address autoconfiguration or manual configurations), the host then obtains configuration information from a DHCPv6 server, such as the addresses of DNS recursive name servers. This procedure is known as a DHCPv6 stateless configuration. There is a weakness in using a stateless DHCPv6 service; if hosts generate some predictable addresses, attackers can easy easily reduce the address searching space. In contrast, if the node obtains both addresses and other configuration settings from the DHCPv6 server, this process is known as a stateful DHCPv6 service. If a DHCPv6 server is configured to use the stateful address configuration mechanism, network administrators can either assign addresses sequentially from a specific range or generate addresses randomly. The former can be leveraged by attackers to reduce the address searching space when performing IPv6 address-scanning attacks, whereas the latter generates stable addresses that do not follow any specific pattern.

Table 3.1: Sample of sub field identifiers IID allocation mechanism

IPv6 Address	Host Name
2001:200:0:2::800:1	lo-0.hitach2.nara.wide.ad.jp
2001:200:0:2::800:3	lo-0.juniper4.nara.wide.ad.jp
2001:200:0:2::1800:1	hitachi1.otemachi.wide.ad.jp
2001:200:0:2::1800:5	lo-1.foundry6.otenachi.wide.ad.jp

3.2.1.3 Manually-configured addresses

Instead of using the mechanisms above, some studies [107, 108, 113] have discovered that manually configured IID mechanisms are popularly used for allocating IPv6 addresses

for routers or servers. For this, network administrators choose one of the following patterns to assign an address to a new server.

Sequential increase host numbering The Interface ID field is sequentially increased when generating a new IPv6 address. In addition, the IID field contains few non-zero bytes, the remaining bytes are all set to 0. For example 2001:df0:0::1, 2001:df0:0::2. In this scenario, the search space for this pattern can easily be reduced.

Use bit fields as subfield identifiers Some network administrators allocate four non-zero bytes to the Interface ID field, so as to classify different groups in the same subnet. For example, table 3.1 shows that the first two bytes from the Interface ID field indicate the group ID, and the low-order two bytes identify hosts in this group.

Embedded-IPv4 scheme Each field of an IID contains 16-bit values and is represented by four hexadecimal digits. However, some studies have shown that network administrators often encode one byte of the IPv4 address into each 16-bit field of the IID, for instance, if a host had an IPv4 address '194.109.20.106' in the current network, a network administrator wants to move this host in to a IPv6 network (2001:888:0::/48) with the subnet ID 0x24, the new IPv6 address for this host looks like 2001:888:0:24:194:109:20:106 or 2001:888:0:24:194.109.20.106.

Service-port Some network administrators put the service port into the lowest-order 16-bit field or the second lowest-order 16-bit field to indicate which service this port is running on, for example, 2001:db8::1:80 or 2001:db8::80:1 refer to the server that opens port 80. In addition, Chown et al. [28] mention that service ports are sometimes encoded in hexadecimal notation, such as 2001:db8::1:50. As we mentioned before, if the IID field contains some predictable patterns, it can be leveraged by attackers to reduce the IPv6 address search space. Table 3.2 summarises the existing IID allocation mechanisms by comparing their visibility and security. The former implies the mechanism has predictable patterns. The latter evaluates the difficulty of reducing the IPv6's address search space for each mechanism.

3.2.2 Interface ID al location mechanisms Usage Survey

Building from the previous studies [28, 108, 107, 113], we have launched a passive measurement survey from our UoA (University of Auckland) IPv6 network. We collected our data from a link connecting a UoA IPv6 network with IPv6 networks outside UoA. We observed 72,931 traffic flows per hour. The traffic flows were reasonably high between 9am-11pm, 105,269 traffic flows are found in this period and the traffic rate drops to 30732 flows during 0am-9am. We built a high-speed flow monitoring system which is able to process and record the first nine packets of each flow into a pcap file every hour. We collected our samples from both flow directions in the period from 2014-05-09 to 2014-08-09. Figure 3.3 shows the logical diagram of our data capture environment.

Table 3.2: Summary of IID allocation mechanisms

IID allocation mechanisms	Description	Visibility	Security
Small-Integer	This scheme is easily identifiable; we verify the Small-integer scheme sets most bytes in the IID to be 0. It can be easily recognized by checking the number of zero bytes in the IID field, for instance 2001:1318:100c:1::1. Additionally, the values in the IID field have been sequentially increased when generating a new IPv6 address	High	Low
EUI-64	This scheme is generated using the EUI-64 algorithm based on each MAC address. The address can be recognized by observing the FF:FE bytes in the IID field	High	Low
Embedded IPv4/ Port	encodes an IPv4 address or port number in the lowest-64 bits of the IPv6 address	High	Low
Randomized	This refers to the temporary [110] and randomised stable [111] allocation mechanisms	Low	High

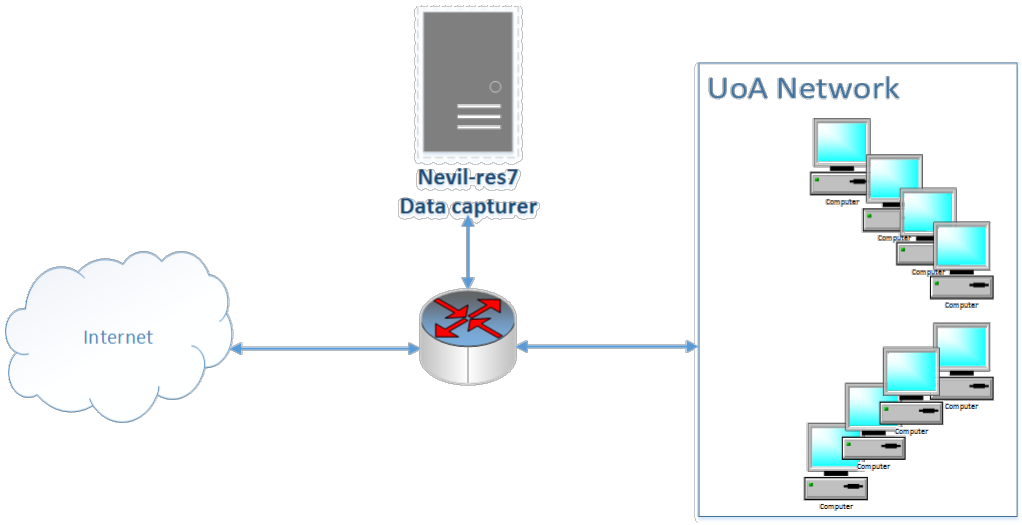

Figure 3.3: Logical network diagram for IPv6 data collection

In this survey, we used several methods to distinguish the servers and clients, the regions, the IID allocation mechanisms and the randomized IPv6 IID values.

Distinguishing IPv6 clients and servers

Trace files do not intuitively tell us whether an address in a packet is a 'client' or 'server'. Therefore, we proposed some methods to classify IPv6 servers and clients. For example, we used port numbers, SYN flags and looked at the DNS reverse zones.

Grouping IPv6 clients into regions

Unlike the IPv4 protocol, IPv6 has divided addresses into two parts: the upper 64-bit network prefix gives information as to a node's location; the lower 64-bit IID field contains information about how an IPv6 node has been configured. Our 'regions' are our own university (UoA), and geographic regions covered by the Regional Internet Registries (RIR): APNIC, ARIN and RIPE [103].

Identifying IID allocation schemes

Identifying IPv6 IID allocation scheme, we developed a python program to distinguish the IID allocation schemes from results, for instance, if the IID field contains 0xFFFE, we put this IPv6 address into the EUI-64 category. If the IPv6 address looks like the Embedded IPv4 scheme, the program will convert the last 64 bits into the IPv4 format, and try to ping this address, if the host responds to an ICMP echo request, we assume that this address uses the Embedded IPv4 scheme. For the randomized scheme, we designed two steps to verify our results, in the first step, the program checks the values in the IID field, if the field is 64 bits long and no significant patterns can be detected, we counted addresses in the randomized category. In the second step, we use a frequency distribution plot to examine the IIDs we counted as 'randomised' and checked the continuous probability function among IID values. In Figure 3.4 the x axis represents the range 0-2^{64}, while the y axis indicates the number of occurrences for groups of IID values, each bar in the plot shows the frequency of occurrence of a given IID value. We observe that such distributions spread more or less evenly across the 0-2^{64} range, with no obvious gaps or significant spikes. However, there is a 1 in 2^{16} chance of a randomized IID being misclassified as an EUI-64, because of the 16 bits 'FF:FE', in addition, if the universal bit has been configured, the probability of a pseudo-random IID being misclassified as EUI-64 is 1 in 2^{17} and is ignored. Our EUI-64 misclassified results are similar to results from Plonka et al. [1] in 2015.

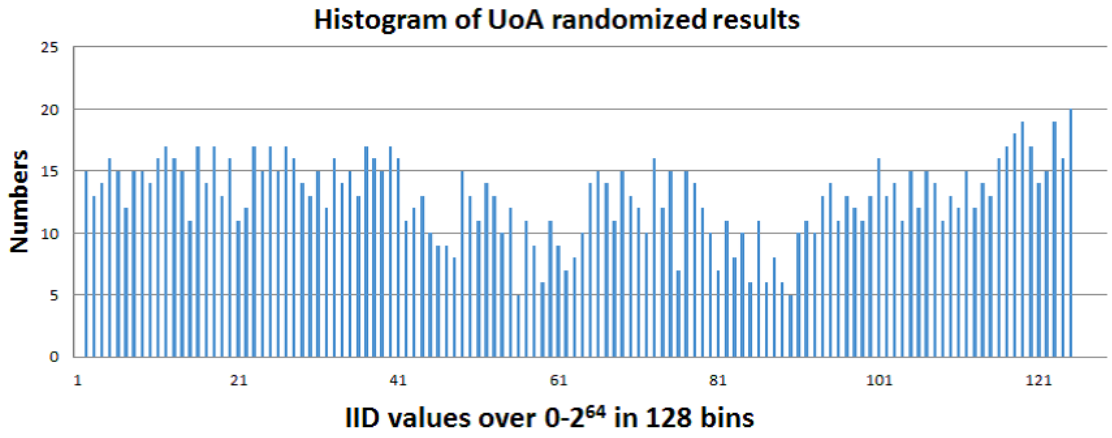

Figure 3.4: Histogram (Randomized IID schemes)

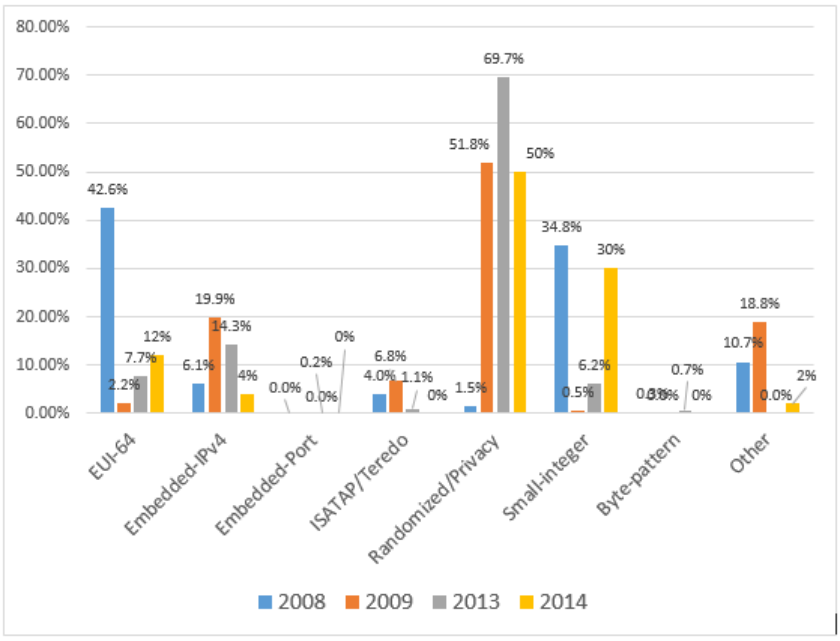

Figure 3.5: IPv6 Client IID Address Allocation Mechanisms Usage Timeline

Figure 3.5 compares IID allocation usages of IPv6 client address in past five years (2009-2014). The horizontal axis indicates the numbers of different IID allocation schemes and the vertical axis shows the percentage for each allocation scheme. In 2008, Karpilovsky et al. [107] examined the ratios of various IID allocation mechanisms. They found that in April 2009 49.5% of IPv6 addresses were the existing IPv4 address embedded into a new IPv6 address (the Embedded-IPv4 scheme); but five months later, the data had been updated and only 6.1% of users were using addresses that were generated by the Embedded-IPv4 scheme. In the same year, Malone [108] conducted a similar experiment to study the IPv6 address usage on different servers. For instance, he found that 70% of routers used the small integer allocation mechanism, and 5% of routers used the Embedded IPv4 scheme. One year later, Shen et al. [113] collected IPv6 traffic from a major ISP in China for one month. Compared with Karpilovsky's results, the ratio for random IPv6 addresses was sharply increased: they observed 51.8% of addresses were using the randomized scheme. They consider that the difference between the two results is caused by the different observation time-frames and the prevalence of Windows hosts in China. The Embedded IPv4 scheme remained the same and the auto-configured mechanism was not widely used in China. Again, in 2014, Gont and Chown [28] analyzed IPv6 addresses obtained from web servers, nameservers, mailservers and clients in 2013. Some interesting trends emerge in their results. At the client side, 69.7% of clients use random addresses, however Gont and Chown still observed a significant amount of exposure to EUI-64 based addresses (14.31%). The client results from our survey are similar to those in [28]. Although our data are taken from a different time and location to the data in [28], our results are quite consistent with those in that study. The largest proportion of IPv6 addresses seen are generated by using a random IID allocation mechanism. In order

to make sure our results were correct, we applied a frequency distribution plot to examine the IIDs we counted as 'randomised' and checked the continuous probability function among IID values. The results imply that most network administrators do care about security and privacy and therefore use unpredictable values in the IID field. The next most common technique seems to be the small-integer mechanism, 30% of these addresses allocate only a few bytes to the IID field while setting unallocated bytes to zero. Some EUI-64 addresses are observed: in particular, UoA contributes a large proportion of the uses of the EUI-64 mechanism. After further investigation, we found that some faculties at UoA use the EUI-64 auto-configuration mechanism to generate a global IPv6 address for a Lab computer to access other IPv6 networks, because this strategy can help them to manage the network more easily. When we compare the results observed in 2009, there is an absence of the use of Teredo, ISATAP and 6to4 allocation techniques in our results, as well as a decrease in the use of embedded IPv4 address in the IID field, which suggests that in some areas network administrators have changed their IID allocation strategies from transition mechanisms to other solutions. In contrast, the manual IID allocation mechanisms are still commonly used for allocating IIDs for IPv6 servers. We presume the manual mechanism assists in the easy management of the network and identification of faulty servers.

3.3 How feasible it is to launch DNS reconnaissance attacks against IPv6 networks?

We mentioned earlier that Van Dijk [26] discovered a new searching mechanism to

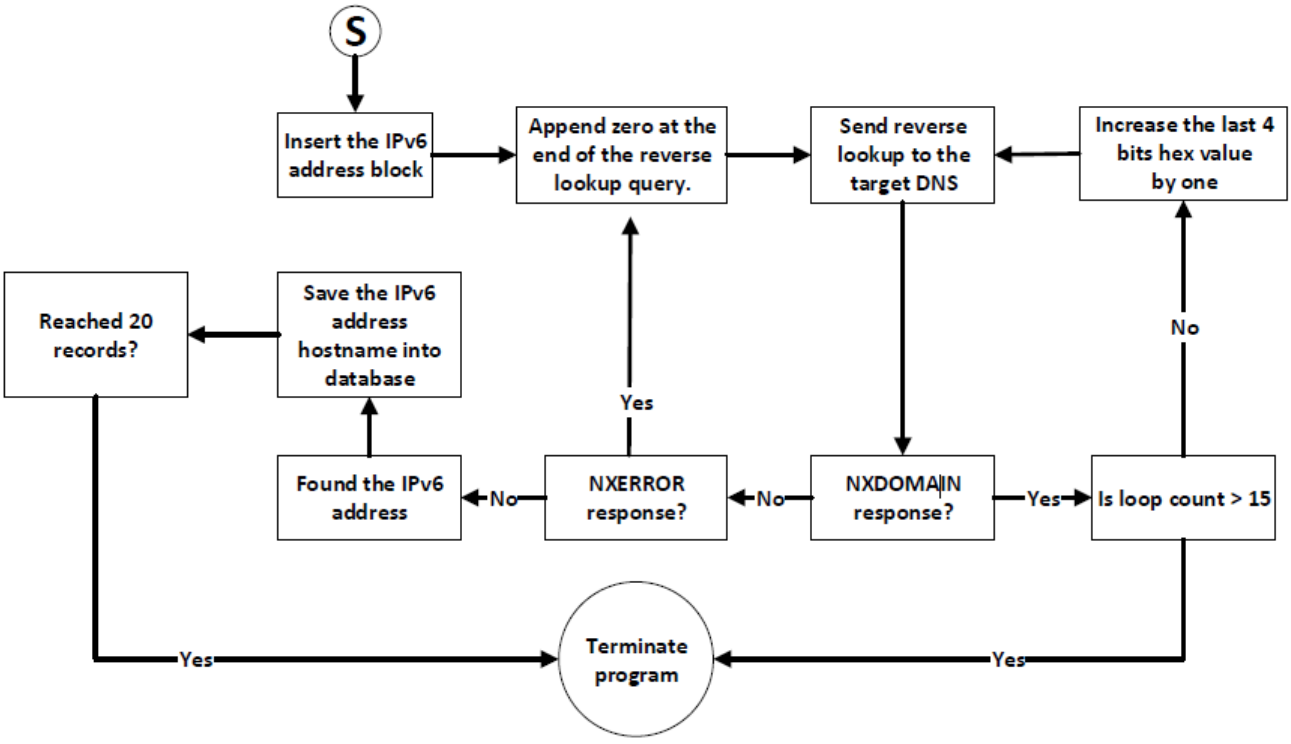

Figure 3.6: Flow diagram of detecting IPv6 reconnaissance attack

In this survey, we use the Regional Internet Registries Statistics website [103] to select the target network prefix and DNS servers; this website updates the deployment status of the IPv6 network regularly. We pass the network prefix of the target domain and the DNS server address to the program, the program analyzes the network prefix and prepends a new nibble (all the new nibbles start with value zero) to the given domain address. The program then sends a reverse lookup with this new address block to the target DNS server. For example, if an input address prefix is '2001:620:0/48', the program creates a reverse lookup query '0.0.0.0.0.0.2.6.0.1.0.0.2.ip6.arpa' and sends it to the DNS server. If the DNS server response is 'NXDOMAIN', the program will check the current nibble value. If the current nibble is equal to value 'F', the program will be terminated; otherwise it increases the current (low-order) nibble value by one. In contrast, if the 'NXERROR' response comes back, it will create a new nibble to add to the existing IPv6 address and this new nibble starts with zero. We limited our searching to a maximum of 20 records for each IPv6 network prefix.

Table 3.3: Columns show the number of IPv6 domains in early 2014 observed in top 20 countries

Country	**Assigned ::/48 network prefixes**	**The number of surveyed network prefixes have more than 20 live IPv6 records**
United States	832	74
Sweden	197	47
Germany	180	34
United Kingdom	107	28
Russian Federation	108	23
Australia	194	21
Netherlands	124	21
Czech Republic	73	19
France	64	19
Ukraine	65	16
Poland	94	15
Switzerland	69	15
Austria	85	15
Brazil	185	14
Indonesia	122	14
Argentina	52	14
Canada	115	12
Belgium	26	11
Norway	40	11
Slovenia	47	10

We launched our survey from January 2014 to March 2014 and collected results from fifty countries that have a significant IPv6 deployment. The survey results show that it

is possible to discover IPv6 addresses from most surveyed DNS reverse zones. Table 3.3 shows the top 20 countries from our results that have a large number of deployed /48 address blocks and the number of surveyed network prefixes have more than 20 live IPv6 records.

The reason for choosing /48 network blocks is that Hinden et al. [114] recommend RIRs assign /48 address blocks to each registered organisation. Therefore, to have a complete IPv6 address, network administrators need only allocate the Subnet (16 bits) and Interface ID (64 bits) fields. From our survey results, we have observed that some DNS reverse zones can be leveraged for gathering the IPv6 hosts results. In addition, some network administrators have used predictable patterns for allocating addresses for IPv6 clients and servers. In principle, we thought that if a country has a large number of assigned IPv6 address prefixes, it should also have more DNS domains that map those prefixes. However, the results from the top fifty countries indicate that some countries have many assigned address prefixes with few IPv6 records in DNS reverse zones. For example, India had 80 assigned IPv6 address prefixes, but no DNS records have been detected from the target DNS servers. There are several known possible explanations for this, such as, some countries are still in the early stages of deploying IPv6, so they do not yet have the IPv6 domains information in their DNS servers, or they haven't set up reverse DNS domains in the IPv6 networks or they configured some security mechanisms for preventing reverse zone searching, such as configuring wildcard records for the entire range of IPv6 clients (Section 2.2) in [115]. If the domain hasn't set up an 'IP6.arpa' zone or used wildcard records, our DNS reverse search method will fail to gather IPv6 addresses from the DNS server.

3.4 Conclusion

In this section, we have discussed the possibilities of launching an address scanning attack in a remote IPv6 network. We briefly explained why traditional address scanning attacks are less feasible in an IPv6 network and reviewed some new IPv6 address scanning techniques that have been proposed from the previous studies, such as reducing the address searching space by observing the predictable patterns from the IID field and gathering IPv6 host addresses from the DNS reverse zone. We have tested some existing reconnaissance tools in IPv6 network for launching IPv6 reconnaissance attacks, none of them are able to do "brute-force" address-scanning attacks in an IPv6 network. Based on our experiences, we listed two tools that can be used to launch IPv6 reconnaissance attacks.

- THC ToolKits: Dnsrevenum6.c gains IPv6 addresses from the 'IPv6.arpa' zone
- IPv6toolkit-v2.0: Scan6, this option specifies the target address prefix/range of the address scan. The program sequentially increase the nibble and send the ICMPv6 message

The detailed discussion of IPv6 address configuration and IID allocation mechanisms has

been covered in Section 3.2. Our survey results in Section 3.3 demonstrated the allocation usage changes in past five years, for instance we observed that, since 2013, IPv6 client addresses are increasingly being assigned by a randomized IID allocation mechanism, and we noticed that some network administrators are aware of the need to use some non-predictable patterns for IPv6 clients. Our results clearly show that many network administrators and operating system vendors have put increasing emphasis on security and privacy concerns when they allocate an IPv6 client's IID field. In contrast, some network administrators still allocate their Interface ID fields with predictable values for clients and servers. We remark that network administrators prefer to use meaningful values in the Interface ID field perhaps to help them identify a host machine when something goes wrong, for example, some network administrators use an existing IPv4 address in their Interface ID fields. It is important that network administrators should avoid having predictable values in their IID fields.

The DNS reconnaissance survey in Section 3.3 shows some potential issues in current DNS reverse zone deployment. For example, some network administrators save the IPv6 client addresses in the DNS reverse zone without any protection. This leads to a security issue in that attackers can probe the DNS reverse zone to obtain the IPv6 client information. Second, if the IPv6 host addresses are using some predictable patterns, it will help the attackers to reduce the time needed for finding other hosts in the same network.

Table 3.4: Requirements for the applicability of network reconnaissance techniques

Reconnaissance techniques	**Requirement**
Launching reverse lookup Search	• Network administrators create an ip6.arpa zone • Every IPv6 host has a record in the ip6.arpa zone • The firewall does not block the reverse lookup query
Reducing the subnet IID search space	• Network administrators use sequential numbers to allocate IPv6 addresses • Network administrators use a simple pattern to allocate IPv6 addresses • Network administrators do not use random numbers to allocate IPv6 addresses

In this chapter, we demonstrated how feasible it is to launch an effective network

scanning attack in the existing IPv6; we did this by doing two surveys. However, there are some pre-conditions of launching those reconnaissance techniques in the existing IPv6 networks. We have listed few requirements in table 3.4.

Based on our findings, we have to find some secure solutions to detect the DNS reconnaissance attack in IPv6 networks. Many security solutions have been deployed in the existing networks. Different solutions come with different features for protecting the existing network architectures. In our study, a security solution for detecting IPv6 reconnaissance attack must ensure the following requirements:

- Must be able to inspect contents of high-speed streams of packets
- No changes for the existing protocols
- Can be deployed as a network-based or host-based solution

Based on our requirements, we analyzed the most common security solutions in the existing network. We notice that some existing security mechanisms have limitations for achieving our goals, such as some are host based only solutions, others cannot inspect the contents of data packets or we have to change the existing protocols. In contrast, an IDS can be used at either network or host based environment. It monitors and detects intrusion activities and events from the packet payload. We will discuss different detection techniques in IDSs in the next chapter, the detection approach advantages and disadvantages are also covered in Chapter 4.

Chapter 4

Defences against network attacks

This section gives an introduction to IDSs, including how IDSs monitor and analyse user and system activity, and how to perform a statistical analysis of activity patterns based on matching them to known attacks. We begin with a discussion about different types of IDS, such as Network Intrusion Detection Systems (NIDS) that perform an analysis of passing traffic in an entire network or subnet: Host Intrusion Detection Systems (HIDS) which monitor your existing system and match user activities to previously identified malicious behaviour. We then discuss different intrusion detection approaches. Signature detection identifies attacks based on the incremental knowledge obtained from previous attacks. In contrast, anomaly detection refers to 'normal behaviour based' detection, it searches for attacks by identifying all unusual actions (abnormally high CPU load, unauthorized login, etc). The remainder of this chapter is organised as follows.

Section 4.1 gives an overview of IDSs, we briefly explain the main features of IDSs.

Section 4.2 discusses IDS architectural variations, we describe the differences between network based IDSs and host based IDSs. In addition, we consider the requirements for configuring both IDS solutions.

Section 4.3 explains IDS detection approaches and related works, we compare the strength and weakness of using the signature based IDSs and the anomaly based IDSs.

Section 4.4 gives our findings and conclusion

Due to advances in Internet technologies, computer networks and other information technology solutions bring convenience and efficiency in our daily life, our society; economy and industry have become highly reliant upon them. However, over the past decades, Internet and computer systems have raised numerous security issues. The Symantec report 2016 indicates that the number of zero-day vulnerabilities increased 125% from the year before. Around 100 million fake technical support scams have been blocked by Symantec. An average of one million web attacks was blocked each day in 2015, an increase of 117 percent (more than double) compared with 2014 [116]. Therefore, it is important to find an effective way to protect our networks. There are some existing solutions on the market, such as firewalls that can filter inbound network traffic, antivirus software used to stop worms and similar malware, authentication requirements that introduce an access

control mechanism and VPNs that encrypt dataflow between headquarters and agencies over the Internet. However, some mechanisms have limitations in detecting attacks from the internal network, while others cannot inspect the contents of data packets. It is important to put in a second line of defence; an IDS helps network administrators to detect unwanted traffic passing through a network or being forwarded to a particular device. The IDS can be applied at either software or hardware level. It monitors and detects intrusion activities or events, for instance illegal and malicious traffic or traffic that violates a security policy. In the following section, we start with an introduction to IDS, then move to discuss two IDS architectures and how they differ. Finally, we discuss different detection techniques in IDSs, the detection approach advantages and disadvantages are covered in the conclusion.

4.1 Overview of Intrusion Detection System

In general, an IDS provides three essential security functions: it monitors, detects, and responds to unauthorized activity by insiders and outsider intrusion. In the intrusion detection procedure, IDS monitors the traffic or events occurring in a computer system or network, then scans packets for patterns that match a pre-defined detection mechanism set (such as signatures, rules or scripts). Each detection mechanism contains information of a known vulnerability, threat or pre-attack probe. If an IDS detects any threats then it sends an alarm to the network administrator. Most IDS platforms allow users to create a custom signature base that can add a new attack into the existing detection set. Below we summarise some main features that are provided by IDS:

- Monitoring and analysis of network and system activity
- Assessing the integrity of packet contents
- Evaluating packet patterns based on matching them to known attacks
- Abnormal activity analysis

In current networks, IDS is broadly classified into two categories based on where it is located: A NIDS or a HIDS.

4.2 Architectural variations

4.2.1 Network Intrusion Detection System (NIDS) Component Types

An NIDS is often used with a 'tap' (port mirror) configuration, it analyses network traffic that passes from a physical layer to an application layer. If any unwanted or malicious events are detected in a particular layer, the system raises an alarm. Most NIDS solutions are simple to deploy on a network and can monitor all traffic that reaches the network.

There are two types of NIDS solution; an NIDS appliance solution requires a vendor to provide an operating system, software solutions and hardware whereas a software solution

contains the NIDS software and the OS only. The chief advantage of using a software-only NIDS is that it is less expensive than an appliance-based NIDS. However, if users choose the software solution, they need to provide the hardware and to configure each piece of hardware; the compatibility of hardware and software may become a problem, because some software may not be supported by a particular hardware component.

The NIDS solution involves four physical components:

Sensors monitor network traffic and make decisions based on whether a traffic flow contains suspicious activities. Deciding where to locate the sensor is important when deploying an NIDS solution. Normally, multiple sensors are deployed at specific points around the network. For example, a sensor can be located near the firewalls, switches or routers.

A management server acts as an analyser; a management server collects all the results from the different sensors. The management server can make final decisions based on what the sensors have reported.

Database servers save all the events from a sensor or a management server. The database server provides information for network administrators to track network disasters and identify attacks.

A console provides an NIDS user interface; a network administrator can configure or access the NIDS via the console. The console can be installed as a local program on the administrator's computer or on a secure web application portal.

4.2.2 Host Intrusion Detection System (HIDS) Component Types

Unlike the NIDS, a HIDS locates sensors in servers or workstations to detect attacks on an end host. In this design, the HIDS not only monitors network traffic on an end host, but also detects system activities. The system makes decisions based on its default settings. The HIDS inherits some existing NIDS components, but it adds some new features. In this solution, a sensor is located on numerous host types, such as server hosts, client hosts and application servers. A server is a computer that allows clients to make a connection to send and receive data: for instance, Web, email or File Transfer Protocol (FTP) servers. A client is a user's computer that establishes a connection to the server. An application service is a program that runs on the server. Like the NIDS solution, the sensor aims to detect malicious behaviours that are occurring on a particular host or that try to access that host. An event log is generated if any malicious activities are observed. Both NIDS and HIDS have their own strengths and weakness, Table 4.1 lists advantages and disadvantages separately.

Table 4.1: HIDS features vs NIDS features

	Host based Intrusion Detection Systems	**Network based Intrusion Detection System**
Advantages	**Monitors System Activities**: A host based IDS can detect all system activities by analysing the operating system logs or events. It can monitor user logon and logoff activity, when the user connect to the network, etc. **Detects attacks that a network based IDS fail to detect**: Host based systems can be very useful in protecting attacks from internal users, such as If an unauthorized user makes changes to system files. The host based IDS can detect it by analysing the system logs **Does not require additional hardware**: Host based IDSs are installed on the host, so there is no additional hardware requirement	**Monitors Network Activities**: A network based IDS is deployed at the edge of the network, it will listen for any attack on the target network regardless of the type of operating system the target host is running. In addition, NIDS can stop the attacks before they can get to a host and so compromise the system **Detects attacks that a host based IDS fail to detect**: A network based IDS is normally deployed outside the firewall, therefore, it can detect malicious attacks that have been rejected by the firewall. This information can be very useful for security analysis **Does not require additional changes**: Network based IDSs are deployed independently and do not affect existing systems or infrastructure. The network-based IDS systems monitor all traffic at the edge of the target network
Disadvantages	• High false alarm rates • Consume host resources • Host based IDSs do not see rejected attacks that never hit the host inside the firewall	• High false alarm rates • Cannot detect attacks within encrypted traffics Packet loss under high loads • A network-based system cannot give the detailed information about system activities • The NIDS often used with a 'tap', so it require additional hardware to be configured as a port mirror

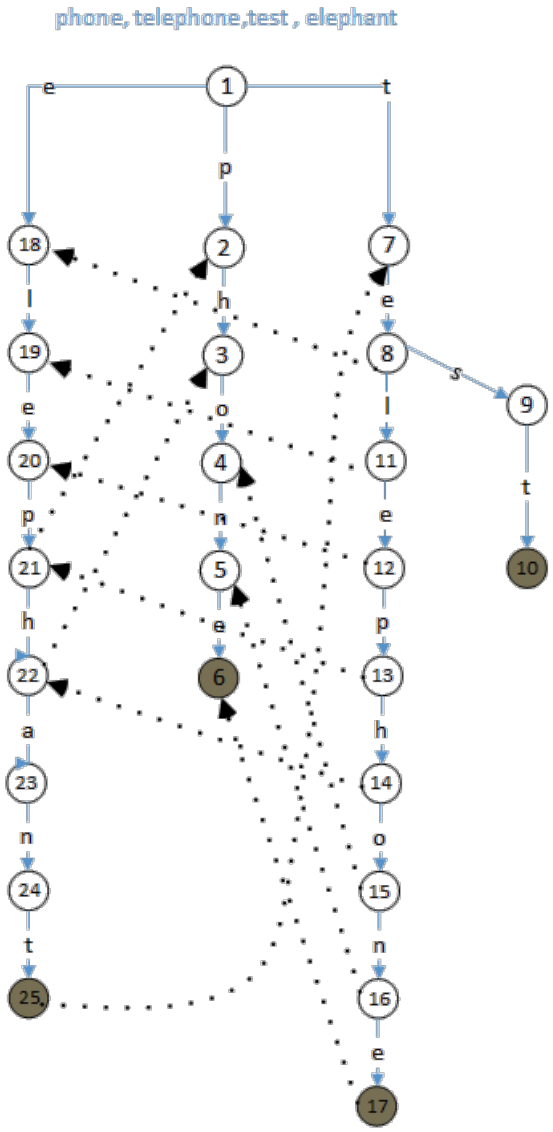

Figure 4.1: Using Aho-Corasick algorithm to process the input strings (from [8])

4.3 IDS detection approaches and related works

Based on the detection approaches, the IDSs are divided into two categories: Signature based solutions and Anomaly based solutions.

4.3.1 Signature-based intrusion detection

This solution requires users to configure a set of rules that includes a list of attack signatures. Each signature contains some significant patterns learned from previous attacks. The IDS monitors packets on the network and compares them to a defined signature or to attributes of previous malicious threats. If a match is detected, an alarm is raised. In most IDS solutions, users are allowed to define their own signatures; they can specify a particular port to be monitored or a special pattern to be checked in the packet payload. This information helps the IDS to identify suspicious traffic flows. The packet header rules store 5-tuple information for each packet; a content inspection rule consists of a string or regular expression pattern from previous attacks. Traditionally, an IDS inspects packets

deeply by scanning every byte of the packet. Naturally, several improvements have been proposed in the last two decades. Our study shows the two searching algorithms that have been used most widely in the current IDS solutions.

Aho-Corasick algorithm

Aho et al. [117] proposed a simple and efficient algorithm in 1975, some existing IDS solutions (Snort and Suricata) still use it as the default pattern searching algorithm. In this approach, they use a pattern-matching machine to represent a pre-defined language as a set of strings; users can test whether an input string matches any set of the given strings. The pattern state machine processes an input text string and is composed of three functions: a [goto] function, a [failure] function and an [output] function:

- A [goto] function constructs a goto graph; the goto graph starts with a root node that represents a state, 1. Each input keyword is entered into a subsequent node. A search starts from state 1 and a path through the graph spells out a keyword. If no failure is detected during the search, the matched keyword will be passed to an output function
- A [failure] function is triggered when the [goto] function reports failure, for example, if a current input character is not found in the current node or the subnodes on the same path, the pattern-matching machine will call the [failure] function to search alternative paths for processing the character
- An [output] function merges duplicated output states into a new output state

Figure 4.1 illustrates how to use the Aho-Corasick algorithm to build a goto graph for an input string list {phone, telephone, test, elephant}. The goto function constructs four paths to spell out these keywords; the solid lines are the normal path for each keyword, while the dotted ones are failure transitions. For example if we use these graphs to process an input text string 'uelephonetest', the machine starts from a root status and reads an input symbol 'u', because the existing state does not have a record for the symbol 'u'; therefore, the machine repeats a cycle at its current state and reads the next input symbol 'e'. The record for that symbol is detected and the next state is state 18, because g(0,e)=18. The state transition keeps moving on this path until reaching state 25. The machine reads an input symbol 'e' and makes two state transitions in this operating cycle. Since g(25,e)=fail, the machine finds an alternative at state 7 and processes the rest of the string on this new path. The Aho-Corasick algorithm is the earliest solution and is widely used for signature-based IDS. For example, Snort's detection engine uses an optimized version of the Aho-Corasick algorithm. Snort uses a two-dimensional chain for pattern matching. The horizontal axis stores some common attributes; once common information is detected, it will start from the closest pattern strings. In addition, some solutions try to enhance the searching speed of the Aho-Corasick algorithm. For example, in [118], Lo et al. propose a solution to enhance the hardware string-matching performance speed and to reduce memory usage by applying a bitmap and path compression to the Aho-Corasick algorithm. A different approach has been proposed by Qu et al. [119], they made a hybrid

solution by combing the multithreading and paralleling IDS approaches together. In this approach, each detection agent holds a small and unique rule set, the incoming packets have been broadcast to all the agents. All agents access the same packet with its own rule set; the alarm is raised if any match is detected. This new solution reduces the IDS processing time and improves detection performance.

Regular expression signatures

A regular-expression mechanism is another signature-matching algorithm; it uses character classes, unions, optional elements, and closures to enhance a signature-based NIDS's flexibility. In addition, it improves searching efficiency by adding effective schemes to perform pattern matching. A normal regular expression can be represented by a finite state automaton. In [120], Hopcroft et al. introduce two finite state automata: a Deterministic Finite Automaton (DFA) and a Non-deterministic Finite Automaton (NFA). The DFA takes input symbols into a transition function and gains a single next state from the function. Instead of returning a single next state, the NFA solution returns a set of states. Most studies show that NFAs are compact but slow; DFAs are fast but may require more memory while processing. In the past ten years, most studies focus on making DFA more efficient, such as [121], where Gong et al. reduced the construction time, memory and matching time by using a multi-dimensional finite automaton in the original DFA model.

4.3.2 Anomaly detection techniques

Signature based solutions address and stop some common attacks in the current network. However they are not capable of detecting novel attacks. In contrast, an anomaly intrusion detection system uses the normal behaviour patterns to identify an intrusion. In general there are two steps for starting an anomaly detection system. The first step builds the normal behaviour patterns from statistical measures of the system features; in the second step the anomaly IDS compares the normal behaviour patterns to detect any abnormal traffic pattern and determine the presence of intrusive activities. A number of anomaly detection mechanisms have been proposed for detecting deviating activities. Existing solutions are categorized into statistical methods, machine learning-based methods and data-mining methods. A brief description of each method is given in the following paragraphs.

Statistical anomaly detection

In a statistical method, users specify a target server as a subject and create two profiles for it: a current profile and a stored profile. The stored profile is generated by network administrators before the deployment, it contains an activity intensity measure, an audit record distribution measure, categorical measures and ordinal measures. The system periodically compares the current profiles with the stored profiles and looks for a difference. The statistical approach can be categorized as model based or non-model based. The model based statistical IDS generates the normal behaviour modes for specific aspects of the network, any deviation from the defined modes are deemed as anomaly activities. In contrast, if network behaviours cannot be characterized by a model, the

non-model based approach is applied. One of the widely used model based approaches is Principal Component Analysis (PCA) [122], In [123], Mechtri et al. proposed a PCA based IDS to improve the operational efficiency and lower use of system resources by transforming a relatively large number of network connections into a smaller number of input data vectors. The results demonstrated significant improvements of processing speed with a lower CPU usage.

Machine-learning-based anomaly detection

A machine-learning solution allows a system to learn normal behaviours from system-to-system or user-to-system interactions. Existing machine-learning approaches can be divided into two categories. Artificial Intelligence techniques introduce statistical modelling to handle symbolic knowledge for determining discrepancies between normal and anomalous activities. Computational Intelligence methods refer to nature-inspired methods; they are used to solve complex problems where no previous information can be referred to. Some previous studies have explored how to use the machine-learning algorithmic method to generate more accurate alarm results. Beaver et al. [124] used in-situ learning and semi-supervised learning strategies to build an intrusion detection model, the system learned both normal and attack traffic, to make a decision on whether an event is malicious or not based on this information. Tian et al. [125] proposed another method based on a neural network and the Particle Swarm Optimization (PSO) algorithm along with Rough Set. The rough set helps an artificial neural network to select and input attribute subsets and the PSO optimized these selected parameters for ANN. Both proposed methods demonstrated higher detecting and recognition accuracy. Data-mining-based anomaly detection Some researchers have focussed on data-mining algorithms that are used to reduce the false alarm rate. This approach creates bounds for accessing valid network activities and sets the security levels for monitoring attack activities in daily traffic. A variety of classification techniques have been introduced in the data-mining approach, but there are some common steps that need to be applied:

- Data collection phase: captures all network flows and saves the information to the relevant database. This information includes source and destination IP address, source/destination port, protocol, number of bytes, service type and TCP flags
- Data filtering and pre-processing: allows users to define the data filtering rules or configure the pre-processing level. The user can configure the system to check only relevant information, such as the information in the TCP or UDP headers
- Mining phase: In this step the system uses the pre-defined rules or learned model to classify the unknown data samples

In [126], Chawla et al. use the Synthetic Minority Over-Sampling Technique (SMOTE) to classify misuse detection and apply Nearest Neighbour (NN), Density-Based Local Outliers for anomaly detection. The results show that using more than one detection scheme can improve prediction accuracy. In this section, we discussed three anomaly detection techniques: In the statistical based IDS, the behaviours of the normal system

are learned before the deployment. In contrast, the data mining based IDS captures the claimed behaviour from available system data. Finally, the machine learning techniques on explicit or implicit models enable the patterns analysed to be categorized. The comparison of all the three techniques is shown in Table 4.2.

Table 4.2: Comparison of All the three anomaly techniques

Technique	**Advantages**	**Disadvantages**
Statistical based	Generate new profiles by accumulating the observed behaviours during a short period	Parameters and metrics are very difficult to set. Easily influenced, could be trained by attackers
Machine learning based	Reducing false alarm rate and maintaining detection accuracy	High operation cost of learning the normal behaviours; high resource consumption
Data mining based	Detect intrusions without prior knowledge	Difficult and time-consuming availability for high-quality knowledge/data

4.4 Conclusion

Security issues are growing every year, therefore deploying an effective and secure solution becomes increasingly important. In this chapter, we presented an overview of the use NIDS of and HIDS in network security solutions. These systems are able to generate alarms, if any anomalous behaviours are observed. The host based intrusion detection monitors or analyses system log files and detects malicious or intrusive activities in a single machine. The network based IDS monitors traffic between different network components on the same wire. It can connect using a network tap or a switch port that mirrors traffic. We also explained the signature based and anomaly based IDSs based on the design architectures, the pros and cons of two IDS approaches are shown in table 4.3. IDSs introduce several features to detect internal and external attackers, provide a real-time report, trace user activity and diagnose the impact and assist the security management of your system. However, there are still some challenges to using IDSs.

Most IDS are still signature based solutions and work on attack signatures. The signature database needs to be updated whenever a different kind of attack is observed. Furthermore, in order to reduce the false alarm rate, once an attack is detected and reported, human intervention must be involved to determine how it occurred, correct the problem and take necessary action to prevent the occurrence of the same attack in future

IDSs are not a solution to all security concerns, they act as one element of an existing security architecture that monitors and reports all intruder attempts

While deploying an IDS solution to monitor HTTPS or HTTP 2.0 traffic, it is important to keep in mind that the current IDS solution cannot monitor and access encrypted

Table 4.3: Comparison of the two IDS approaches

	Signature-based IDS	**Anomaly-based IDS**
Advantages	• Simplest and easiest solution to deploy for detecting known attacks • The system generates a detailed contextual analysis report that makes it easier for network administrators to identify and detect the attack • Low false-alarm rate	• Effective to detect new attacks • Network administrators can update the new profiles in the real time • Network administrators generate the customized profiles that avoid attackers to test what will trigger an alarm
Disadvantages	• Easy to bypass the detection by using new ways to launch attacks • High operation cost to analyse each signature • No novel attacks can be detected as IDS can only analyse the known attacks	• High false alarm rate • Difficultly of defining profiles • Additional cost of adding new hardware

traffic. We learned some alternative solutions from the previous studies [127, 119], for instance, 1) Decrypting traffic before passing it to an IDS. 2) Using HIDS solutions instead of NIDSs.

With different vendors working on eliminating the shortcomings of Intrusion Detection Systems in the last decade, more and more efficient IDS tools have been introduced. These systems will have a certain degree of success in detecting most existing security attacks. However, different vendors have different design ideas and use different detection algorithms for identifying malicious behaviours. In the next section, we will explain the similarities and differences among three IDSs based on the followed factors: design architectures, detection procedures and the rule format.

Chapter 5

Overview of Snort, Bro and Suricata

In the previous section, we described an intrusion detection system. Network administrators use this system to monitor network or system activities so as to detect malicious behaviours or policy violations. We discussed the strengths and weakness of using signature based and anomaly based solutions. The former relies on spotting a duplication of attacks that have been detected before, in contrast, anomaly solutions can be used to detect a new attack, but their false alarm rate maybe high. In this chapter, we start with three popular open-source IDS tools: Snort[1], Bro[2] and Suricata[3]. All three tools are widely deployed in the existing IPv4 networks; many organisations use one of them to protect their networks. The use of an open-source tool offers two significant benefits. First, the flexibility of the open-source tool is a major advantage; we can easily modify the source code to detect new attacks. Second, the open-source tool is available free of cost. The remainder of this chapter is organised as follows:

Section 5.1 gives an introduction of the design goals for three IDSs

Section 5.2 compares design architectures for three IDSs

Section 5.3 compares data capture algorithms for three IDSs

Section 5.4 compares packet detection algorithms for three IDSs

Section 5.5 compares rule formats for three IDSs

Section 5.6 gives conclusion

5.1 Introduction of Snort, Bro and Suricata

This section gives an overview of three popular IDSs, we introduce the general information for each tool followed by some primary design goals.

5.1.1 What is Snort?

Snort is a signature based detection open-source tool that provides both network intrusion detection and network intrusion mitigation; it comes with a set of relevant rules and features that help users to detect attacks and probes. For instance, based on the

[1]https://www.snort.org
[2]https://www.bro.org
[3]http://suricata-ids.org

user's specification, Snort can detect buffer overflows, port scans and web application attacks, etc. In addition, users can specify new rules to detect a new attack and configure rule actions to either drop or log malicious packets.

5.1.2 What is Bro?

Bro is an open-source IDS that passively monitors network traffic and looks for suspicious activity. Bro uses a specialized policy language that help users to monitor and analyse attacks: for instance, Bro provides default policies for detecting SSH brute-forcing and validating Secure Sockets Layer (SSL) certificate chains. Again, it allows users to create a new policy for identifying a new attack. Unlike the current signature based IDSs, Bro provides more features for detection of semantic misuse, anomaly detection and behaviour analysis.

5.1.3 What is Suricata?

Suricata is another open-source network intrusion detection system. While it shares some similarities with Snort and Bro, for instance it inherits both pattern matching and script solutions to withstand attacks against it, it introduces a multi-thread packet processing structure to accelerate its operation speed in high-volume networks. Each of the three IDSs come with different primary design goals, some of which are listed here:

5.1.4 Snort design goals

- Has an open source structure. Unlike some commercial network-based intrusion detection tools (Cisco secure intrusion detection system, CyberSafe centrax and Network ice blackice defender), Snort allows users to add their own signatures into the existing rule base. Once a new signature is created and enabled on a system, Snort will immediately apply the new signature in its intrusion detection process
- Supports passive traps. In general, network administrators are aware which services are available on their network. Therefore, they can specify Snort rules to watch for traffic that tries to interact with non-existent services. If any incoming packets attempt to call non-used ports or services, a log or alert will be generated. For instance, if a network is not using File Transfer Protocol (FTP) service, users can configure a Snort rule to raise an 'FTP Probe' alert if they detect that the packet intends to connect to port 21. The key word 'any' is used to specify any IP addresses or ports. A Snort rule to detect the FTP probe attack is shown below

  Alert tcp any any -> any 21 (msg: 'FTP probe!'; sid:2002; rev:1;)

5.1.5 Bro design goals

- Enhancing the speed of operation for high-speed and larger volume monitoring. These days network throughput has increased from Mb/s to Gb/s. Any IDS needs to capture packets as quickly as possible. If any packets are dropped on the monitored

link, that may defeat the purpose of monitoring by missing some important information that identifies network intruders. Bro accommodates such high-performance settings by supporting scalable load-balancing. Large sites typically run 'Bro Clusters' in which a high-speed frontend load-balancer distributes the traffic across an appropriate number of backend PCs, all running dedicated Bro instances on their individual traffic slices

- Real-time notification. Some attacks happen very quickly. Bro allows users to detect the attack in real time; it helps administrators to trace back the attacker much more easily. Because of this, it can minimise the damage and prevent further break-ins
- Flexibility and extensibility. In order to process a high volume of traffic more efficiently, Bro separates the detection mechanism from the policy for rules. It uses different mechanisms to apply different policies for filtering, monitoring and responding to different types of traffic. Again, Bro allows users to add a new policy to detect a new attack. Furthermore, its event handle is treated as a link between its event engine layer and policy script layers. For instance, if a user adds a new protocol analyser in the event engine, the user also needs to write an event handle to process the event generated by this new analyser

5.1.6 Suricata design goals

- Improved performance. Suricata is designed to use multi-threaded processing and aims to balance traffic loads by distributing the traffic over every processor in a Suricata host
- Protocol identification. Unlike some IDS systems, Suricata allows users to define either the protocol type or the particular port in the rule file. In addition, Suricata provides a larger number of keywords that match on protocol fields
- Pattern-based and script-based detection. Similarly to Snort, Suricata uses a pattern-matching mechanism in the default setting; it compares packet information with pre-defined rules. If any matches are detected, an alert is generated. However, the pattern-matching only tests the relevant rules for each incoming packet; there is not an obvious deliberate way to check for pattern relationships among the packets in the same flow. In order to detect this attack, users need to compare the previous information with the current content, which is not possible using the pattern-matching mechanism. Instead of using pattern matching approaches, Suricata introduces script-based detection and well-designed data structures for parsing and saving flow information.

5.2 Comparison of design architectures

In the previous section, we discussed IDSs with different design goals. Therefore, different packet processing flows are used in different IDSs. In this section, we will compare the similarities and differences among three IDSs based on the design architectures.

5.2.1 Snort design architecture

Figure 5.1 demonstrates the main components and the packet processing steps in the Snort system; each component plays a different role and associates with the next component in the Snort architecture. In the initial step, the packet decoder captures a data packet from the network card, which then passes through the pre-processor component, the pre-processor modules normalize protocol headers, detect anomalies, reassemble packets and track the TCP or UDP session. If any anomalies are detected, the pre-processor raises the alerts before the detection engine processing. The detection engine is a core component in Snort; it is designed to access the packet content and identify any malicious behaviour that matches the detection mechanisms. For each detection mechanism, the user has specified rules for either raising an alert or dropping the packet. All the alerts are saved by the output modules.

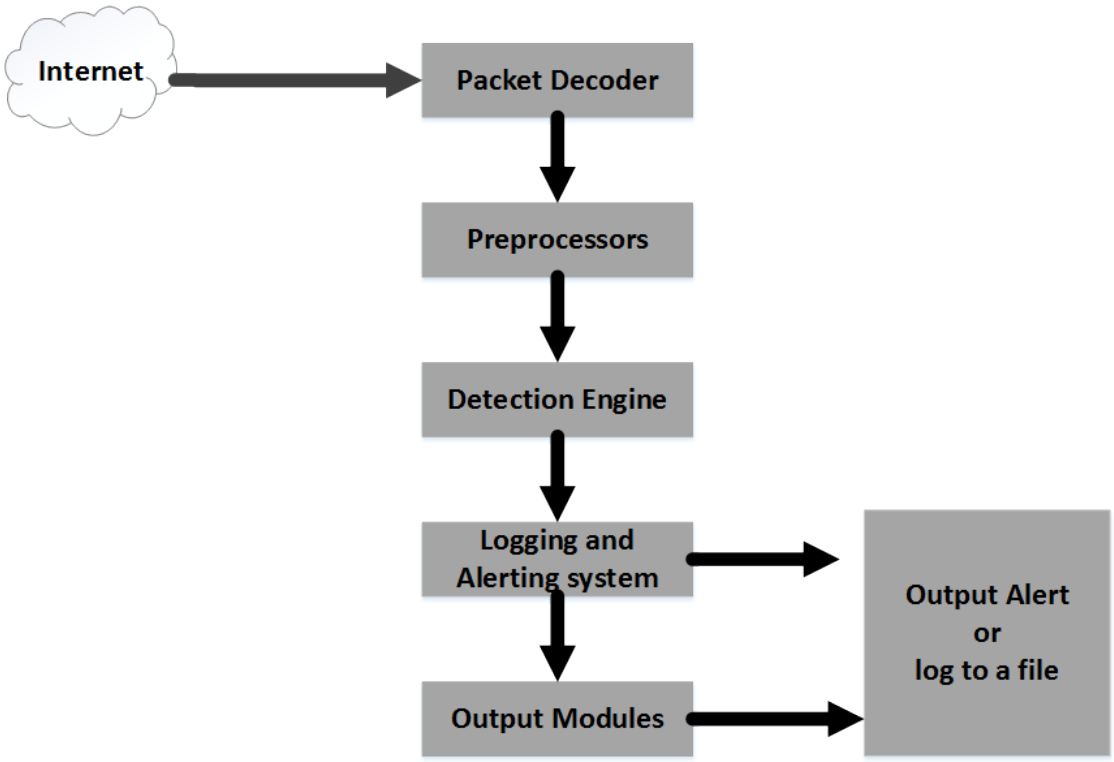

Figure 5.1: Snort architecture and components

5.2.2 Bro design architecture

Snort uses a one direction searching mechanism; it does not allow the component to return the information back to the previous component. In contrast, Bro's design is based on layered structures, as shown in Figure 5.2. It creates bi-directional communication between the event engine layer and policy script interpreter layer. The event engine passes the packet to the policy script interpreter for the detection of any anomalies, and the policy script interpreter returns the results so the event engine can take the right action. The packet processing flows are described as follows: the lowest layer is called the packet capture layer, it uses the libpcap library as a default to get packets

from the monitored wire. The captured packets are passed to the event engine layer, which manages the session states and classifies different protocols. The event engine layer produces an event to describe the relevant network activity and indicate flow information of the network packets. Each event is linked to a corresponding policy script interpreter; as a result, a stream of events is passed to the policy layer. The interpreter accesses the packet behaviours with user-supplied scripts and returns a decision: normal or abnormal network activities. The event engine raises the alert if the malicious behaviours are detected.

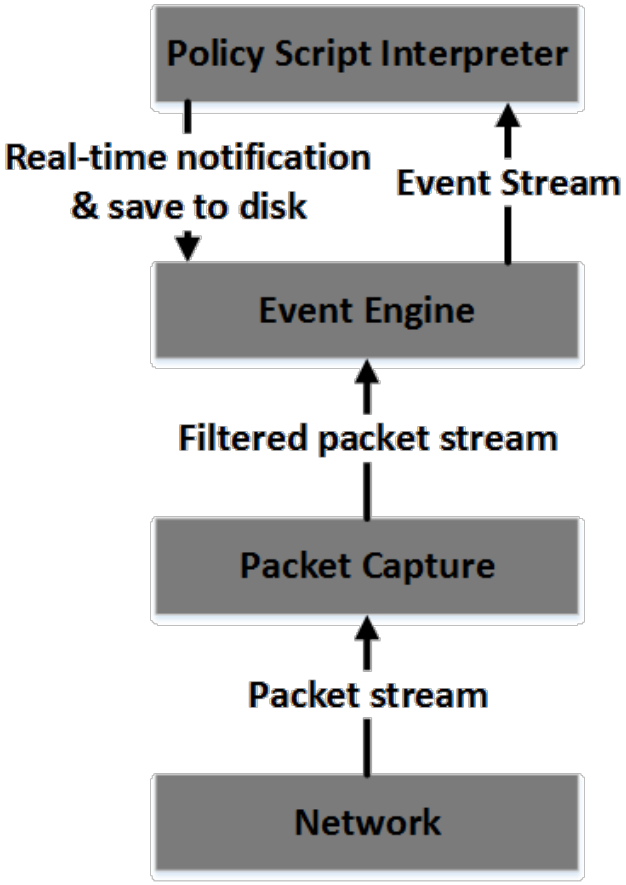

Figure 5.2: Bro architecture and components

5.2.3 Suricata design architecture

Suricata design architecture brings new ideas and technologies to the IDS solutions: it offers a multi-threading solution to improve the flow processing speed. Figure 5.3 gives an example of how Suricata processes incoming traffic with multi-thread detection processors. Suricata collects packets via a packet acquisition module (libpcap) which uses the specified network card to gather the packets and pass them to Suricata. Based on the hardware specification, Suricata then generates a number of threads and runs the different threads on different CPUs. Each thread acts as an instance of packet processing flow for processing multiple packets in parallel, the number of generated threads determines how many packets can be processed in the same time. Each packet is processed by the decoder and stream layer to connect its content to a Suricata support format. The detection

module accesses the packet by loading all signatures and initializing all the detection plugins. After packet analysis is performed, Suricata provides multiple ways to save the alerts.

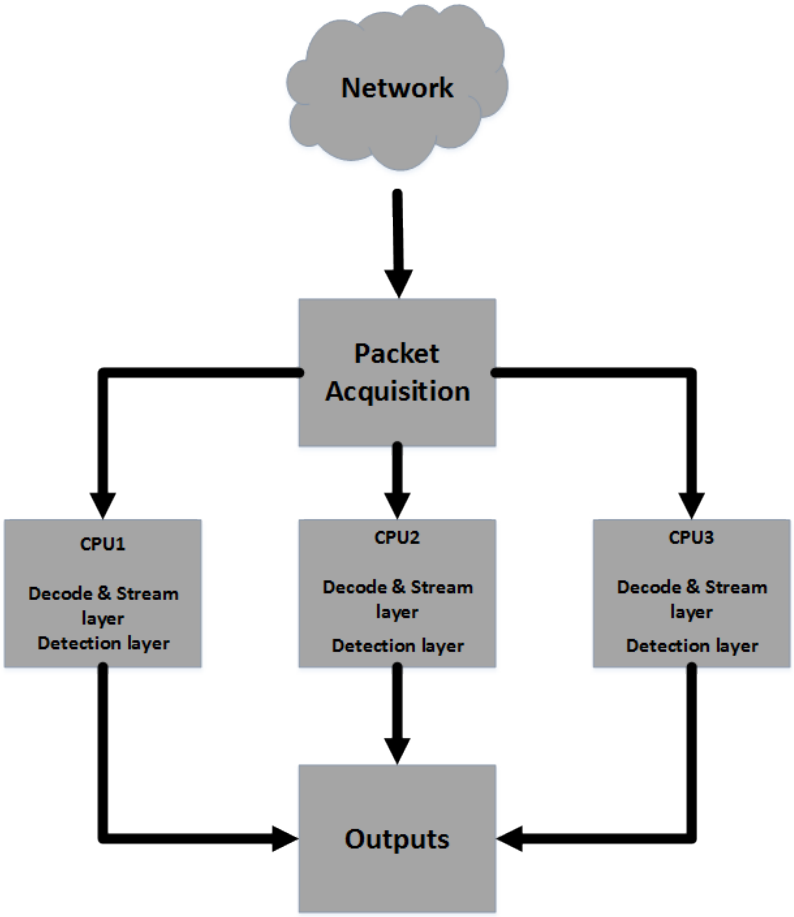

Figure 5.3: Suricata architecture and components

When the packet processing flows for three IDSs are compared based on their design architecture, it is clear that different IDSs employ different methods to achieve their design goals. Snort is a single-threaded system, it processes the packets through all enabled components and no information can be passed back to the previous components. Suricata inherits the basic Snort architecture, such as the packet decoding and the packet detection, but it extends it and provides more flexible solutions and features, with the addition of multi-threading as well as file extraction. Suricata is also compatible with script-based detection. In comparison, Bro offers a full script-based analysis engine for detecting malicious activities via scripts. Suricata is also compatible with the script-based detection. In addition, Bro tracks the flow information by exchanging the information between different layers. In this section, we briefly explained IDSs working flow based on their design architecture. All tools rely on the different components working together to detect particular attacks and to generate output in a configured output channel. In the following sections, we investigate the mechanisms or strategies that are supported by each component.

5.3 Packet capturing mechanism

5.3.1 Libpcap

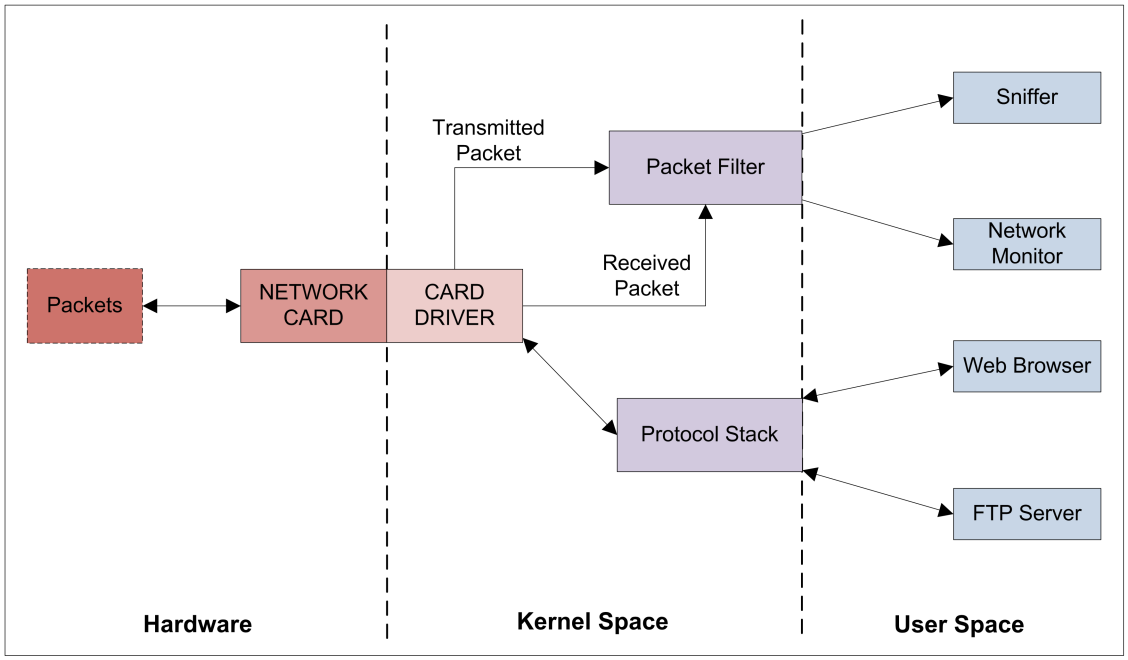

Figure 5.4: Structure of libpcap packet capture (Source from [9])

Libpcap[4] is a hardware-independent open-source library that allows users to sniff the packets from a monitored network. Figure 5.4 demonstrates the packet capture process. In principle, a network card is used to send and receive network packets. The card driver grabs the packets and sends them to the protocol stack. The OS protocol stack analyses the packet and allocates packets to the relevant application. Libpcap is located at the boundary of the kernel space where it can monitor both incoming and outgoing packets from the network interface card. The packet capturing procedure includes three steps:

- Device Initialization. Libpcap allows users to call its pcaplookupdev() function to list all network devices, it then uses getifaddrs() to get their IP addresses and related information. All such network devices are saved in the pcapif list
- Berkeley Packet Filter (BPF) [128]. This provides a filter function for the sniffer, so that it forwards only specific packets. The BPF is called after the driver receives the packets from the network interface
- Packet processing loop. Snort uses a while loop to call the pcapdispatch() function from the libpcap library. pcapdispatch() reads specified packets and passes them to Snort. Snort then uses a PcapProcessPacket() function to process each captured packet based on different protocol types. The packet decoder passes all decoded packets to the pre-processor module for further investigation.

[4]http://www.tcpdump.org/

5.3.2 AFPACKET

AFPACKET is the Linux native network socket. It functions similarly to the memory mapped PCAP, but no external libraries are required. Similar to libpcap, AFPACKET enables the user to configure a memory buffer for captured packets. This means that the memory allocated for the buffer is shared with the capture process, so instead of the kernel sending packets to the capture process, the process can just read the packets from their original memory address. This method saves time and is less consuming in terms of CPU resources.

5.3.3 PFRING

PFRING [129] is another high-throughput Linux kernel module that optimizes load balancing through the ring cluster design. In the packet capturing process, the application copies packets from the NIC to the PFRING circular buffer. Then the applications reads the packets from this circular buffer. The advantage of using PFRING is that it can distribute incoming packets to multiple rings; it allows multiple applications to process packets simultaneously.

To conclude the summary of existing packet capturing mechanisms, we noticed that Bro did not support the AFPACKET capturing mode. While other the two mechanisms are supported by three IDSs, Libpcap is configured as a default mode for three IDSs to sniff the incoming packets. PFRing only works in the 'Bro Clusters' mode and Bro does not support AFPACKET. In contrast, Suricata and Snort can use the PFRing mechanism with the PFRing supported hardware. There are no additional requirements to use the AFPACKET in Snort and Suricata; users can enable the AFPACKET mode through the configuration file or the command line.

5.4 Packet detection mechanism

The detection process is the core of IDS solutions; different tools introduce different mechanisms and perform differently when it comes to their performance, resource consumption and accuracy. In this section, we start with an overview of detection engine processing for the three IDSs followed by the different detection strategies.

5.4.1 Snort Detection Mechanism

Snort introduces two components to assist the packet detection: pre-processors and the detection engine.

The pre-processor. This is a first step of the packet detection process, without the use of the pre-processor component it is quite difficult to match patterns from any packets that are not well formatted. For instance, attackers can fragment a large packet into small packets and split their malicious scripts into fragmented packets; each fragmented message will not contain a significant enough signature to match the relevant rules, but when these fragmented packets are assembled together at the receiver end, attack scripts

can be triggered. The Frag3 pre-processor module, used by Snort, is designed to assemble all fragmented packets back into the original packet. The module then passes the original packet to the detection engine. Again, the detection engine simply searches the packet content to match the user specified signature rules. However, even without restructuring the packet content, an attacker can easily bypass this check. For example, if a user creates a rule to detect an http packet that contains 'scripts/iisadmin', attackers can modify the string to 'scripts/./iisadmin' to escape that check. Attackers can also make this situation even worse by inserting web Uniform Resource Identifiers (URI) in hexadecimal characters or Unicode characters. To solve this potential issue, Snort provides the 'Httpinspect' module for restructuring some HTTP URLs into well-formatted HTTP URLs. All the pre-processors have been compiled during the installation. Users can enable or disable different pre-processors through the 'snort.conf' file. Additionally, users can use the customized pre-processors to detect new attacks.

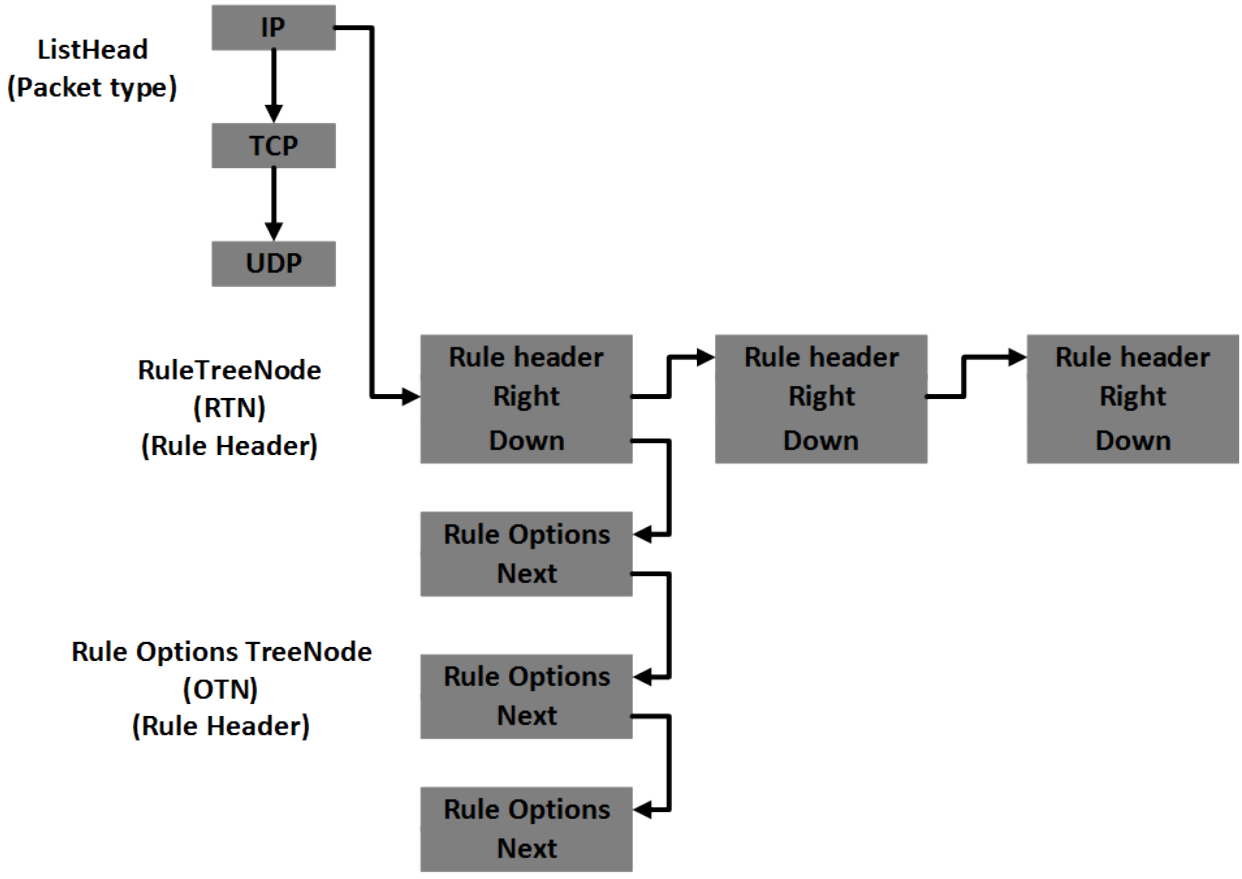

Figure 5.5: Structure of the two-dimensional chain

The detection engine. A core component in Snort is the detection engine; it is designed to access the packet content and find information that matches the relevant rules. Snort implements its pattern-searching mechanism in the detection engine, which reads rules from the user specified rule set and uses the rules to test each incoming packet. If a packet contains information that matches any rule, pre-determined action is taken, such as dropping the packet or triggering alerts. Performance time is a major

concern when implementing the pattern-matching mechanism in a detection engine; low engine performance can cause packet dropping and affect the detection result. Snort introduced a two-dimensional linked list for reading the rule into internal data structures or chains. Figure 5.5 demonstrates a two-dimensional chain example. The horizontal axis is called a Chain Header, it records packet header information, such as IP addresses, port numbers and protocols. The vertical axis is called a Rule Option, it represents a group of rules sharing some common attributes, such as port numbers or IP addresses. Each rule option specifies how to analyse different intrusion behaviours that have been observed from previous attacks.

Snort condenses all the common information into a single chain header and divides detection signatures into different chain option structures to improve the speed of detection processing. For the packet detection process, the detection engine component parses the packet header and matches the information with the chain header; if a packet header match is found, the detection engine starts to compare the payload information with the chain options that belong to the same chain header. The chain options contain attack patterns from previous attacks, if any matching is detected, an alert is raised.

5.4.2 Bro Detection Mechanism

Bro introduces the packet decoding process in the event engine layer, it contains several analysers that check information in a packet header, verify its IP header checksum, sort the incoming packets into the right order based on the connection status, reassemble fragmented packets by using the fragment ID, and classify packets based on different protocols. Bro raises the event based on the packet's behaviours and the generated event triggers the relevant event handle from the policy layer. In the event engine layer, there are two important entities: the Event management and the Protocol analyser. Event Management. Bro uses different events to keep tracking TCP connection status; for instance, by using the TCP flags for each new TCP connection. If the first packet contains a SYN flag, the event engine will set an expiry time for this new flow; a 'connectionattempt' event is generated if the flow has not received the SYN-ACK packets from the destination node but 'tcpattemptdelay' seconds have elapsed. However, if the TCP session is established before that time, the event engine issues a 'connectionestablished' event and deletes the expiry time. In another scenario, if the destination host sends an RST packet to reject the current connection, a 'connectionrejected' event is raised. The 'connectionfinished' event is generated when the event engine receives a FIN packet. There is no connection event for the UDP protocol; Bro uses two events to indicate a UDP sender and a receiver instead. A 'udprequest' event is generated; if the UDP request is detected, a 'udpreply' event is generated for the packets that are sent by a UDP responder. More information on how to use events can be found in the Bro 2.3.1 script reference [130]. Protocol Analyser. The protocol analyser is another core component for accessing different protocols. Figure 5.6 demonstrates the protocol analyser tree structure in Bro. In [10], Larsen explains how to find the right analyser through the Bro data structure. For instance, if a new

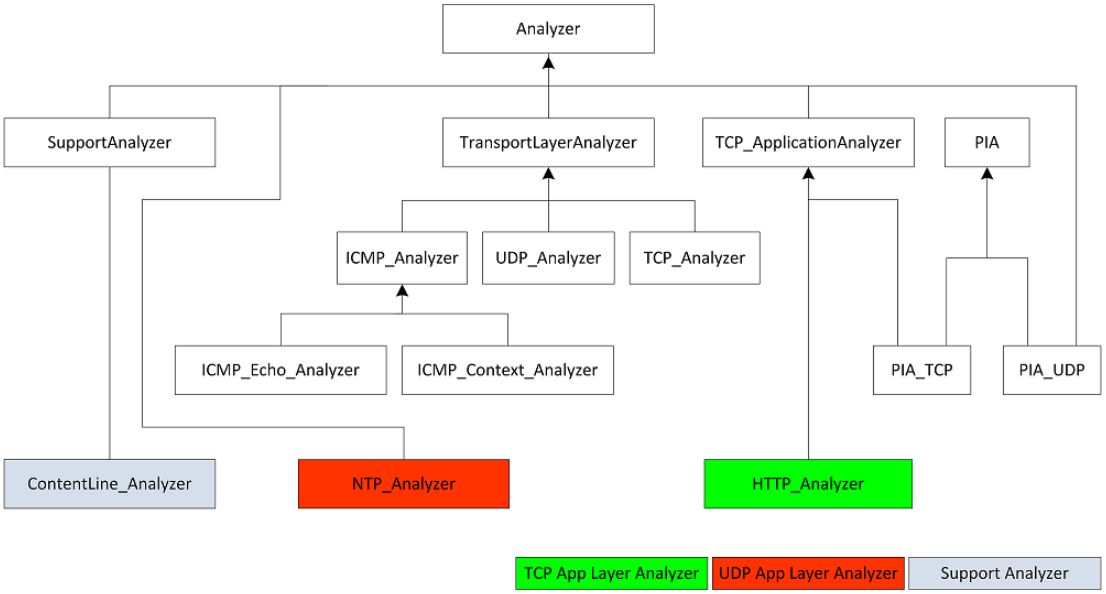

Figure 5.6: Bro analyser tree structure (Source from [10])

connection is detected, Bro creates an instance of this analysing tree to associate this new connection with the correct analyser. He also claimed that "an analyser can support one of two input methods (or both) packet wise or stream-wise. An analyser can accept input via one method (e.g., packet-wise) and pass it on to its children via the other (e.g. stream-wise). The TCP-Analyzer for example reassembles packets into a byte-stream and thus all TCPApplicationAnalyzers only see stream-wise input" [10].

Bro's Policy Layer. The policy script interpreter is located above the event engine layer, different scripts are triggered by different events. Once the event engine generates an event, the top layer of Bro starts to process the event based on the specified policy scripts. A policy script contains server handles, with each handle referencing one relevant event. In other words, any event that has been raised will trigger the calling of a corresponding event handle. Users enable the policy scripts based on their particular requirements. Bro loads all enabled scripts during its initialization processing. If a policy script needs assistance from a disabled analyser, the event engine will automatically enable it. This design allows Bro to only allocate resources that are actually being used. The policy layer only raises an alert if a security breach is found.

5.4.3 Suricata Detection Mechanism

As we mentioned before, Suricata utilises some of the same ideas as Snort, such as the pre-processors or the detection engine processing. However, it modifies the Snort detection mechanism to fit into its multi-threading solution. There are two main differences between Suricata and Snort.

Packet decoding. In Suricata, the decoding pipeline reads packets from a raw socket for the defined capture device in real time; it then decodes the packet based on the Open Systems Interconnection (OSI) model. Its layer 2 is decoded, after which the higher layer' sprotocols are decoded. The decoded data is saved into a Suricata-readable packet

format and passed to the detection module. The detection module accesses the packet with user-defined rules or scripts. Suricata's rule matching is similar to Snort's. Suricata introduces some new keywords to simplify the detection process, such as the keyword 'iprep' for matching IP reputation data, and the 'dnsquery' keyword for analysing DNS responses.

Detection engine. Snort runs on a single thread. If users want to run a multi-threaded solution, they need to run multiple instances of Snort and assign traffic to different instances based on the flow type. Suricata can run multiple detection modules in parallel and one flow can be processed in multiple detection modules at the same time. There is a risk in running the flow in a multi-threading program; for instance, if any malicious packets get split into different threads, the detection engine may be unable to match the complete signature. To solve this problem, Suricata allows the detection engines to share coordination information, but doing so also increases the computation load on the system. Furthermore, network traffic is directed to the relevant flow. The keyword 'memcp' can be used to specify the maximum number of bytes the flowengine will use. This parameter can mitigate the engine overload problem, caused by attackers creating a lot of flows to flood the system.

Both Bro and Snort use a single thread to access the incoming packets, Bro maintains the flow information and allocates different events for packets. Bro's packet detection mechanism will not check all the policy script, the policy script is only triggered when the packet matches a specified event.

Unlike Bro, Snort is a traditional signature based IDS, no flow caching functions are introduced in Snort; it loads the default rules when the Snort instance is launched. The detection engine then analyses the packet and decides the rule set, the packet goes through all the rule sets until the matching is detected. The weakness of using a single-thread IDS is that only one packet can be processed in a detection flow. In contrast, Suricata makes it possible to launch multiple packet detection processors simultaneously, and examine several different packets in parallel.

5.5 Detection rule format

Different IDSs come in a variety of rule formats for detecting suspicious traffic in different ways, in this section, we explain the basic rule formats among three IDSs.

5.5.1 Snort language

Snort rules are expressed in a simple language which can be used across different platforms. It is comprised of three parts: the rule action, the rule header and the rule options. For example, Table 5.1 shows a Snort rule example for detecting the Portable Hypertext Format (PHF) probe attack [131].

alert tcp 192.168.1.2 50051-> 192.168.1.6 80 (content: "/cgi-bin/phf"; msg: "PHF probe!"; sid: 20023; rev:1;)

Table 5.1: Snort Rule Structure

Rule Actions	**Protocols**	**Source addresses**	**Source port**	**Destination addresses**	**Destination port**
Alert	TCP	192.168.1.2	50051	192.168.1.6	80

Table 5.2: Snort header fields and features

Rule protocol	**TCP**	**Snort identifies TCP, UDP, ICMP, IP, ICMPv6 and IPv6 protocols**
IP address	192.168.1.6	The addresses are formed by IPv4 dotted decimal or IPv6 16-bit hex groups and a CIDR [132] block. Users can specify that Snort should look at any address on the monitoring wired link or monitor specific host addresses. In the current version, Snort does not allow users to put the host name in the address field. The key word “any” indicates all valid addresses
Port numbers	80	Snort allows users to specify the port number in various ways, such as any ports, static port definitions (e.g: 23 for telnet, 80 for http), ranges, and by negation (e.g: if we want to log every port except the port range between 5000 to 5100, we could define a rule which looks like log tcp any any -> 192.168.1.0/24 !5000:5100)
Direction operator	-> <>	The direction operator reflects the packet’s path; the left side is a source node, and the right side is a destination node. <> is a bidirectional operator that tells Snort to analyse in both directions

Table 5.3: Snort rule option

Content	"/cgi-bin/phf"	The value in the content field helps Snort to do string pattern checking, if the packet has the same information, the defined rule action is taken
Msg	"PHF probe!"	The plaintext string gives more information about the anomalous activities
Sid	20023	A unique ID for identifying the rule. Every rule has a unique sid. User-defined rules should have a higher sid than 10,000
rev	1	Revision number ID

The rule action tells Snort what to do when malicious behaviours are observed. For instance, the alert action allows Snort to raise an alert and store the packet information in the default log path. The rule header helps Snort to decide where a packet comes from and which protocol a packet is using. Table 5.2 gives the name and the function description for each field in the Snort rule header. A rule option is comprised of two parts: a keyword and an argument. The keyword tells Snort where to search. For instance, the keyword 'content' tells Snort to look at the packet payload and call a pattern match function. The 'nocase' keyword allows Snort to ignore upper or lower case in the packet payload. The argument is an input parameter that is passed to the pattern-matching function. Users can define one or more options and use a semicolon to separate each option. Table 5.3 gives the name and the function description for each field in the Snort rule option.

5.5.2 Bro language

The original goal in designing Bro's language was to avoid simple mistakes. Its designers proposed various features to achieve this goal. For example, unlike some existing NIDS solutions that use interpreted languages, Bro detects inconsistent types at compile-time and makes sure that all variables reference a valid type at run-time. Bro creates a way to express IP addresses, port numbers and time directly. Furthermore, Bro avoids the drawback inherent in using the null termination symbol. This problem is that the real content of the string could be subverted by extracting only the content before the null terminating symbol and the content following the null terminator might be ignored. For instance, if a user sends a FTP request "USER user\0 USER root", an IDS that uses the NULL termination mechanism to extract a string will find only 'USER user' and miss the string after '\0'. The result of missing that information might be to subvert the target host or the network. Bro does not use the null termination symbol as a string terminator. Some traditional variable types have been re-used in Bro, such as int, count, double, bool and string, but Bro also introduces some new types. Table 5.4 shows some new data types introduced by the Bro language. More detailed information can be found in [130].

Table 5.4: New data types introduced by Bro's language

Data type	**Description**
Time	Bro provides a few built-in functions to show an absolute time. The 'current_time' function returns a time of the operating system; 'network_time' reflects the time the last packet was processed
interval	Interval is a time interval (seconds) equal to the time between two time values
pattern	Bro allows users to do pattern searching with regular expression patterns. The pattern is constructed by enclosing text within forward slashes(/). Bro provides two pattern-matching mechanisms. One is called 'exact' and the other 'embedded'. 'Exact' matching checks whether the input string exactly matches the pattern 'Embedded' matching checks whether the input string includes the pattern or not
port	Users are allowed to define the port number followed by a transport-layer protocol type, for example: [80/tcp] = WWW
event	An event is implemented with an event handler. The event handler can be called through the script, and it never returns a value. There are three different ways to execute event handlers: • In the event engine, the event engine queues an event. The event handler is invoked when the previous events have been moved out from the queue, and the event engine has finished processing the current packet • Directly calling an event handler from a script • Setting a schedule to process an event handler. Bro allows users to invoke an event handle some time in the future
hook	"A hook is another flavor of function that shares characteristics of both a function and an event. They are like events in that many handler bodies can be defined for the same hook identifier and the order of execution can be enforced with priority. They are more like functions in the way they are invoked/called, because, unlike events, their execution is immediate and they do not get scheduled through an event queue. Also, a unique feature of a hook is that a given hook handler body can short-circuit the execution of remaining hook handlers simply by exiting from the body as a result of a break statement (as opposed to a return or just reaching the end of the body)" [130]

```
function init (args)
    local needs = {}
    needs["packet"] = tostring(true)
    needs["payload"] = tostring(true)
    return needs
end

-- match if packet and payload both contain HTTP
function match(args)
    pkt = 0
    pay = 0

    for k,v in pairs(args) do
        if tostring(k) == "packet" then
            a = tostring(v)
            if #a > 0 then
                if a:find("HTTP") then
                    pkt = 1
                end
            end
        elseif tostring(k) == "payload" then
            a = tostring(v)
            if #a > 0 then
                if a:find("HTTP") then
                    pay = 1
                end
            end
        end

        if pay == 1 and pkt == 1 then
            return 1
        end
    end

    return 0
end

return 0
```

Figure 5.7: an example of Suricata script language (Lua)

5.5.3 Suricata language

Suricata introduces two detection formats: rule based and script based solutions. Suricata's rules are expressed in a simple and readable structure, which can be used across platforms. Suricata and Snort share some common language features, so Suricata's rules have been made similar to those of Snort. As a result, a Snort defined rule can be reused in Suricata. However, as we mentioned before, a rule based matching mechanism is suitable for attacks in which the pattern does not change very often. If more information is needed about the flow in order to compare the current information with previous information (as happens in some cases), the pattern-matching mechanism will generate a high false rate. In order to solve this problem, Suricata introduced Lua. Lua is a light-weight but very powerful programming language. Lua is similar to Pascal, but with powerful data description constructs. It is frequently used for configuration and scripting. To enable Lua in Suricata, we have to add a new keyword, luajit, in the existing Suricata rule structure. The script name is used as an input argument for this keyword. The following example shows how to launch the lua script from the Suricata rule:
alert HTTP 192.168.2.1 any -> 192.168.2.10 80
(msg: 'LUAJIT HTTP test'; luajit:test.lua; sid:20002;rev 1;)
If any incoming packets match this rule, the script is then loaded from the rules directory and run against a packet. Figure 5.7 presents a Lua script that detects an HTTP flow, the Lua script is comprised of two parts: the init function and the customized function:

The 'init' function allows the user to indicate what buffers the script needs to inspect.

If the buffer is unavailable, the script will not be invoked. The keyword 'needs' refers to a table structure that is initialized through an init: function. In the current Suricata version, the keyword 'packet' refers to the complete raw packet, including protocol headers. 'http.url', 'http.headers' and so on. The 'match' function is used to check whether the incoming packet contains a HTTP request or not. For instance, if the input argument is a packet it then checks the protocol type, if the HTTP protocol is used in the packet the 'pkt' is set to 1, or if the input argument refers to payload it then checks whether the HTTP keyword has been used in the payload or not. The match function will return either 1 if the HTTP request is found, or 0 if not. As stated previously, Snort uses pattern-based rules, which include fingerprints and identifiers existing from previous attacks, Snort's detection engine loads those rules to match known malicious patterns. Bro's language introduces an analysis driven model, it provides a well-designed framework for analysing the incoming packets based on the flow information, saving the packet payload information and updating the flow state information. In this design, Bro is able to support anomaly detection as well as misuse detection. Suricata takes the advantage of Snort and Bro languages, it allows users to use the signature based and script based solution together. It puts the well-known patterns in the rules to reduce false alarms, in addition, Suricata allows users to define customized scripts for detecting new attacks.

5.6 Conclusion

This chapter explored the overview of three IDSs as well as architectures and algorithms to process the incoming packets. Snort is an earlier open source intrusion detection solution and is the most widely deployed IDS worldwide. Suricata is a new IDS solution that brings some ideas from Snort, it introduces a multi-threading solution to accelerate detection speed. Bro is slightly different to Suricata and Snort, focusing instead on network security monitoring, network traffic analysis and high-level sematic analysis at the application layer. After discussing its main design goals, we describe the general design architecture. Like most IDSs, these three IDSs are based on a packet monitor that listens to network traffic at some central point of the network. By closely sniffing incoming packets, Bro can reconstruct the semantics of the connections. For each of them, Bro keeps flow state information. Unlike Bro, Suricata and Snort inspect individual packets. From some of our own experiences while deploying the three IDSs, we found that Bro needed more time to understand and deploy than a system like Snort or Suricata. Furthermore, we demonstrated the similarities and differences of the packet capturing mechanisms, the packet detection processing and the rule formats among three IDSs. Tables 5.5, 5.6 and 5.6 provide an overall comparison between these three IDSs.

Table 5.5: An overall comparison among three IDSs-1

	Snort	**Bro**	**Suricata**
Structure Detection Engine	• Rules form "signatures" • Modular detection elements are combined to form these signatures • Wide range of detection capabilities • Rules system is very flexible, and creation of new rules is relatively simple	Event Engine • Performs several integrity checks to assure that the packet headers are well formed • It looks up the connection state associated with the flow tuple • It then dispatches the packet to a handler for the corresponding connection Policy Script Interpreter • The event engine triggers the policy script interpreter based on the events • logs real-time notifications, recording data to disk or modifying internal state • Adds a new detection mechanism, users need to apply the changes in both protocol analysers and script interpreters	• Rules form "signatures" • Modular detection elements are combined to form these signatures Policy Script Interpreter • The policy script interpreter receives the packet information generated by the detection engine • It then executes scripts written in the Lua language which generates relevant information like logging real-time notifications, recording data to memory or updating the current state • Adds new functionality to Suricata consists of adding a new rule to the rule base and then writing new script handlers in the interpreter

80

Table 5.6: An overall comparison among three IDSs-2

Threads	• Single-thread • Can run the multi-instances with the flow filtering mechanism applied	• Single-thread • Can run multi-instances in Bro Cluster Architecture	• Multi-threads
Design Goals	• Packet sniffing "lightweight" IDS • Libpcap-based sniffing interface • Rules-based detection engine	• High-speed, large volume monitoring • Real time notification • Mechanism separate from policy Extensible	• Multi-threading, hardware capture cards • High level protocol detection (HTTP, etc) • High speed IP matching
Capture methods	• Libpcap, AF_Packet,	• Libpcap and PF_Ring (Bro-Clusters)	• Libpcap, AF_Packet and PF_Ring
IPv6 support	• Enable IPv6 through the command line or the configure file	• Default	• Default
Online/ Offline detection	Yes	Yes	Yes

Table 5.7: An overall comparison among three IDSs-3

	Snort	**Bro**	**Suricata**
Advantages	• Snort can easily be deployed on any node of a network, with minimal disruption to operations • Snort provides tested set of signatures • Portable (Linux, Windows, MacOSX, Solaris, BSD, etc.) • Snort can act as IPS by configuring inline mode	• Bro supports both anomaly-based and misuse-based approaches • Bro-ids is capable to perform application level deep packet inspection • Analysis of file content exchanged over application-layer protocols, include MD5/SHA1 computation for fingerprinting	• Multi-threaded: A multi-threaded architecture allows the engine to take advantage of the multiple core and multiprocessor architectures from the hardware • Supports IP Reputation: By incorporating reputation and signatures into its engine, Suricata can flag traffic from known bad sources
Installation/Configuration	• Snort allows easy enablement or disablement of rules, pre-processors and different matching mechanisms. Users can change the configuration file or type the commands	• Unlike Snort or Suricata, Bro does not have a main configuration file • Instead, users have to modify the code to enable or disable the protocol analyser or particular scripts	• Similar to Snort, Suricata allows users to modify the system configuration from the configuration file or the command line

82

Chapter 6

How effective are IDSs at detecting IPv6 attacks?

In the previous chapter, we have described the features of three IDSs independently, we also compared their data capturing mechanisms, detection processing, and rule formats side by side. It is now time to examine the real world performance of these IDSs, under varying conditions of traffic flow, with varying configurations. The main objective of this is to assess the feasibility of using the popular IDS solutions in high speed and IPv6 networks, both with and without our proposed DNS reconnaissance detection solution. We evaluate the three IDSs using a number of tests. The initial tests were designed to explore the following question: What are the performance ramifications of the three systems when IDS configuration and the amount of network traffic are considered? We also wanted to know how well the IDSs were able to identify a DNS reconnaissance attack. All tests were designed to answer the following questions:

1. What are the key architectural considerations for open source IDSs as network speeds increase?
    - Evaluating different pattern matcher algorithms
    - Evaluating different network packet capture methods
2. How should an open source IDS be designed to more readily facilitate new emerging IPv6 attacks?
    - Detecting IPv6 reconnaissance attacks using default rules
    - Implementing a new detection mechanism in IDSs

In most cases, careful configuration resulted in significant performance improvements. However, none of the IDSs were able to identify a DNS reconnaissance attack using their default rules. We propose and implement a solution in Bro and Suricata. The remainder of this chapter is organised as follows.

Section 6.1 compares the performance of IDS when IDS configuration and the amount network traffic are changed

Section 6.2 provides a proposal for a new DNS reconnaissance detection solution

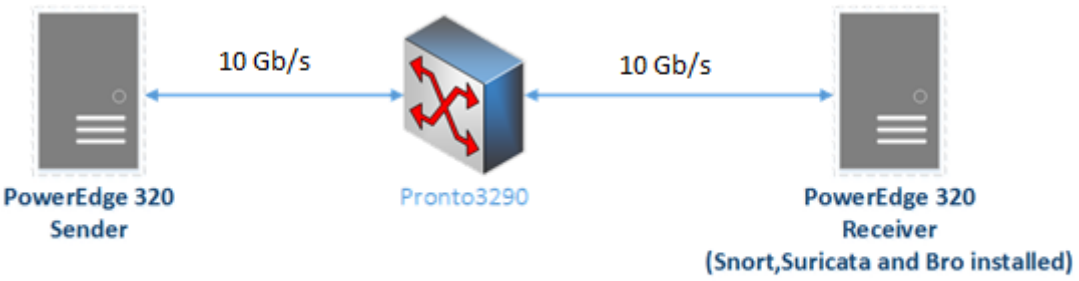

Figure 6.1: Logical network diagram for 10 Gb/s testing environment

Section 6.3 determines the feasibility of implementing the proposed solution in IDSs

Section 6.4 evaluates the DNS reconnaissance detection solution in both experimental and real IPv6 networks

Section 6.5 gives conclusion

6.1 Compare the performance of IDSs

6.1.1 Environment

In this section the testing environment and procedure is described. Test results will be discussed later. We decided to evaluate our tests in two environments: an experiment-testing network and the campus network link. Figure 6.1 shows the logical network diagram for this 10 Gb/s experiment testing environment. In this test network, the infrastructure consisted of two servers and one switch. More detailed information about the test environment can be found in Table 1 (Hardware Specification).

Table 6.1: Hardware specification

Role	Model	CPU	Memory	NIC	OS	IP
Sender	Dell R320	Intel Xeon E5-2407 2.20GHz	16GB	Broadcom BCM57810 10 Gigabit	Ubuntu 12.04	192.168.6.2
Receiver	Dell R320	Intel Xeon E5-2407 2.20GHz	16GB	Broadcom BCM57810 10 Gigabit	Ubuntu 12.04	192.168.6.1
Switch	Pronto P-3290	MPC8541	2GB	Firebolt-3ASIC	XorPlus 1.3	

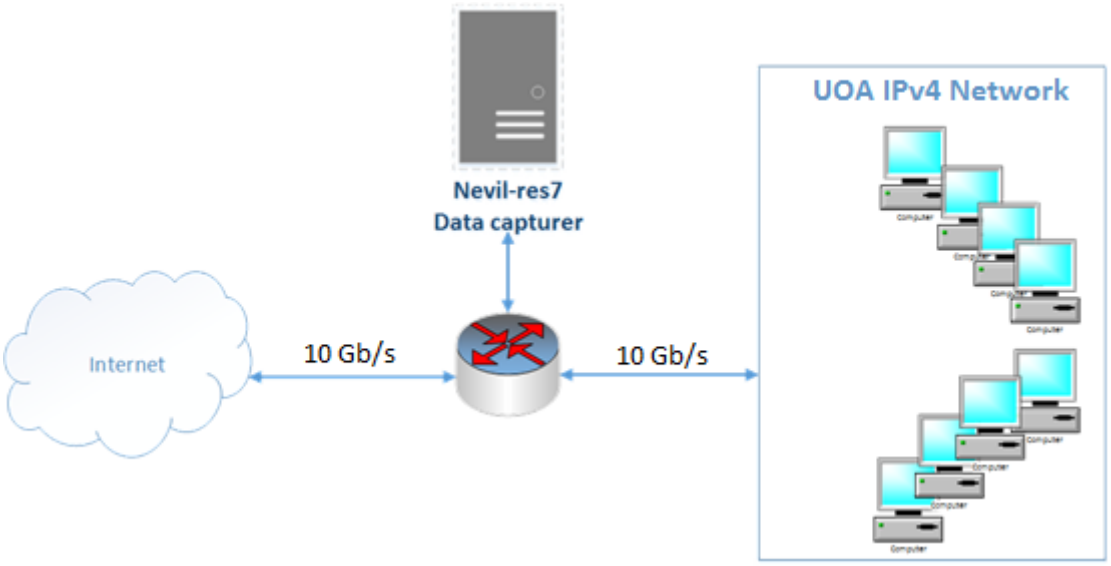

Figure 6.2: Logical network diagram for UoA campus 10G/s link

In the campus network, we set up a port mirror in our campus boundary network, to monitor all the IPv4 and IPv6 traffic. Our campus network setup is shown in Figure 6.2, it has a 10 Gb/s link and the mirror host is a Dell R320 with 32 GB memory.

6.1.2 Methodologies

We have used two methodologies here: in our experiment environment we used the active measurement strategy, we aim to deliver a large volume of data quickly and efficiently through high capacity links (10 Gb/s). However, the standard TCP or UDP implementation cannot be used for these links. Yu et al [133] mentioned some modified implementations which allow GridFTP, Fast Data Transfer (FDT) and UDP-based Data Transfer (UDT) to deliver a high volume of data through high-capacity links, and do a better job than the generic implementations do. The results of their experiment indicate that UDT had some utilization problems when delivering large quantities of data through a 10 Gb/s network. The performance results for GridFTP and FDT are similar to each other, but when they used GridFTP with jumbo frames, it gave better performance for transferring high volumes of data in both congested and uncongested links. Based on the FTP protocol, GridFTP uses parallel data transfer, and improves the performance of high-speed data transfers between hosts over a long fat pipe network. In our experiment environment, we didn't launch any background traffic, instead the Sender (192.168.6.2) uses GridFTP to transfer big data volumes at 10 Gb/s to the Receiver (192.168.6.1). Yu et al's experiments demonstrated that GridFTP was able to handle about twenty flows running in parallel, but the number of dropped packets increased significantly beyond this limit. Therefore, in our experiments, we have evaluated a single flow test case, as well as the twenty flows scenario. In contrast, we used the passive measurement strategy to evaluate the IDS's performance in the UoA campus network link. There are two reasons for choosing the UoA network: we have full view of incoming and outgoing packets, and

the UoA network link provides a typical mix of flow types.

6.1.3 Experiment scenarios

In Chapter 5, we compared the similarities and differences among three IDSs, we found the performance of intrusion detection systems can be affected by the following factors:

1. Software architecture and implementation:
   (a) Detection algorithm optimization
2. Data acquisition (DAQ) methods:
   (a) PCAP
   (b) AFPACKET
   (c) PFRING
3. Detection rules:
   (a) Amount of rules loaded

In this section, all three IDSs were tested in both our experimental and real network environments based on the following scenarios:

- Default IDS configuration
- Optimize each IDS configuration by modifying the detection algorithm: By default, the IDSs come with different pattern matcher algorithms. According to the user manual for Snort and Suricata, different methods offer different performance effects on memory consumption, queue size and CPU usage. We will describe some of the possibilities for improving IDS performance by modifying or configuring the pattern matcher algorithm
- Modify network socket packet capture mechanisms to improve capturing performance. Three popular packet capture methods have been introduced in Chapter 5. Different methods use different architectures to reduce the amount of dropped packets in different network environments. In this section, we try to understand which method is more suitable for our network environment based on the performance results
- Optimize the number of rules loaded. The IDSs come with a general configuration that enables all the rules or scripts. Based on the user's environment, some rules will not be used. Therefore, disabling the unnecessary rules may improve the performance results.

For each scenario, we have measured the following information:

- Number of packets received
- Number of packet dropped
- Average CPU usage
- Memory usage

We have not considered the hard disk read or write usage because, in our tests, we were only writing log information to the hard disk. We verified that the operation cost was very low and that our HDD configuration was fast enough to handle these operations. Therefore, we decided that analysis of the HDD usage was unnecessary.

6.1.4 Experiment results

This section describes the test results. For each test, we considered four performance factors (CPU usage, memory usage, received and dropped packets). A performance baseline is generated from the first experiment scenario. For the following test cases, changes made to the systems will be discussed and we will demonstrate the similarities and differences to the previous measurements. For each scenario, we launched an IDS instance individual for one hour and tested each scenario more than 10 times to get the average results.

- Experiment 1 - Default OS & IDS Configuration: In first test case, all three intrusion detection systems were set up using the default configuration.

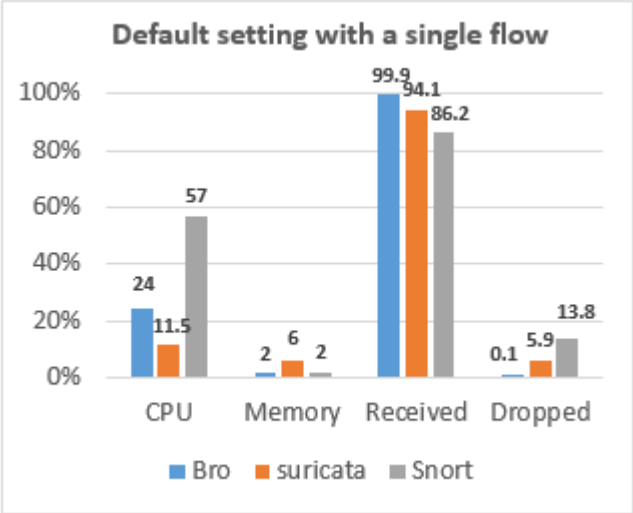

Figure 6.3: Percentage of performance with default IDS configuration with a single flow

Figure 6.3 shows the results of three IDSs in their default configuration, we tested the three IDSs separately to monitor the traffic on a high speed experiment network (10 Gb/s) with a single flow. The three tools generated different performance results. When it came to packet drop rate, Snort dropped 13.8% packets during the one hour testing. Suricata dropped 5.9% and Bro achieved the best drop rate (0.1%). For the CPU usage, Snort used more than 57% of CPU resources, Suricata launched seven threads in two different cores, the average CPU usage was 11.5%. Again, Bro used less CPU than Snort, it utilized 24% of CPU resources.

Figure 6.4 below shows the results of three IDSs in their default configuration monitoring the traffic on a high speed experiment network (10 Gb/s) with twenty flows. In

this test case, the CPU usage for Snort and Bro were increased, in particular, Snort used 93% of CPU resources. For the drop rate, all three IDSs show the default configuration is less efficient for monitoring traffic on a high speed network with multiple flows running in parallel. In this test case, we note that the number of flows can affect the IDS's performance. However, our test environment only generated TCP flows, so in an attempt to better understand how the performance can be affected by the mix of flows, we launched three IDSs in our campus network link with the default configuration.

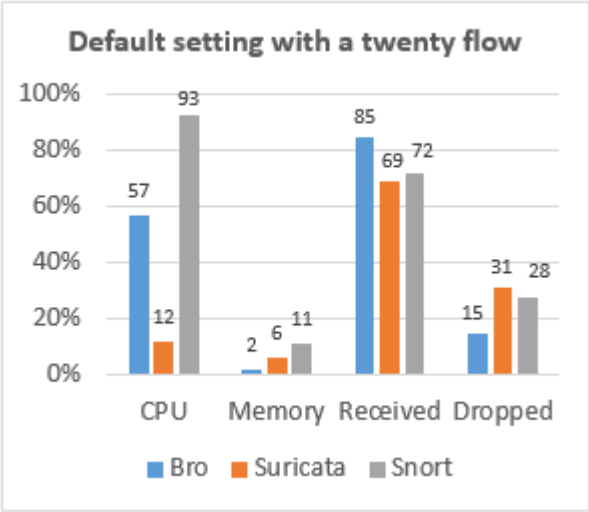

Figure 6.4: Percentage of performance with default IDS configuration with twenty flows

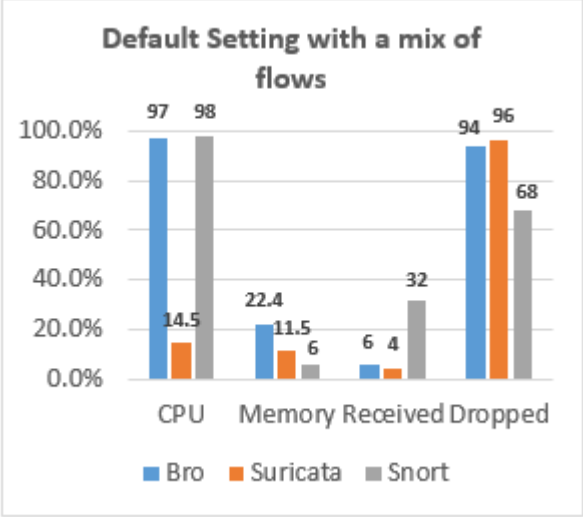

Figure 6.5: Percentage of performance with default IDS configuration with a mix of flows

Figure 6.5 clearly demonstrates that single threaded IDSs created a CPU overload. As the transmission flow increases, the CPU load is also increased, Snort and Bro generate nearly 100% CPU usage. Compared to the other two IDSs, Suricata launched 37 threads across a quad core system, the average CPU usage is 14.5%. When we tested the three

IDSs in our campus network link with mix of flows, the results show that the default configuration is less efficient if used in the high speed network environment with multiple flows running in parallel.

- Experiment 2 - Optimize each IDS configuration by modifying the detection algorithm

In the first test case, we demonstrated the default configuration for IDSs solution is not suitable for monitoring high speed network traffic on the 10 Gb/s campus network link. The CPU usage and drop rate were very high. In this test, we focus on a simple change to the IDS detection algorithm to achieve better results. We derived this knowledge from the IDS's user manual for understanding how to improve performance.

Detection engine pattern matcher algorithm We compared all available detection engine pattern matcher mechanisms in Snort, the results show that 'ac-nq' offers high performance with a low memory consumption in all the test cases. We also evaluated other pattern matcher algorithms that are mentioned in the Snort.conf file.

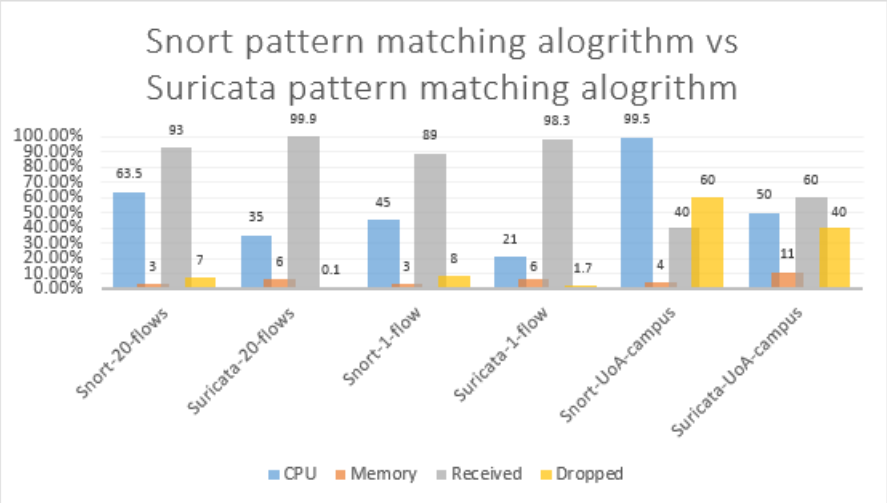

Figure 6.6: Percentage of performance after optimizations

Figure 6.6 demonstrates the performance results of using the 'ac-nq' pattern matcher algorithm in Snort and the 'ac-gfbs' pattern matcher algorithm in Suricata. We tested both algorithms in the following test cases: 1 flow, 20 flows and the UoA campus link.

We evaluated all the pattern matcher algorithms in Snort and Suricata. Some algorithms, although they generated similar performance results, required more CPU resources. A case in point is that both 'ac-bnfa' and 'ac-nq' in Snort captured 99.9% packets, however, the former generated 92% CPU load while the latter only used 65% CPU to process the 20 flows. The memory usage for Snort seems very constant for all test scenarios, we didn't observe significant changes when we configured the different algorithms. The average memory consumption is 3% of the total memory size.

Similar results were observed for Suricata, the default algorithm ('ac') dropped 5% packets when monitoring a single flow. In contrast, the 'ac-gfbs' offers good performance with moderate CPU usage and low memory; the number of dropped packets reduced to 4.9% for the same test case. Considering that other detection algorithms offered a similar drop rate with much higher memory consumption (3-5GiB), it was reasonable for us to choose a low memory usage algorithm.

As we can see above, our different network environments yield different performance results. Configuration choices directly affected the packet drop rate; users can configure different algorithms based on their network environment requirements (hardware, bandwidth and mix of flows).

It is important to note that unlike Snort and Suricata, Bro does not allow the user to modify the pattern matcher algorithm. In addition, when we used the optimized pattern matcher algorithm in the UoA campus network link, Snort reduced its drop rate by 30% and Suricata increased its receive rate by 57% compared to that with the default pattern matcher algorithm. However, the drop rate was still higher than the receive rate, that will affect the detection result. In the next experiment, we tried to evaluate the different DAQ methods with the different pattern matcher algorithms.

- Experiment 3 - Modify DAQ mechanisms to improve capturing performance

AFPACKET is the Linux native network socket. It functions similarly to the memory mapped libpcap, but no external libraries are required. Based on our hardware requirements, we have configured the DAQ buffer memory to 1GiB in order to compare the similarities and differences with libpcap. For improving the processing time and reducing the CPU usage, Suricata introduced the Zero Copy mode for sharing the buffer's memory address with the capture process, so the process can read the packets from their original memory address instead of receiving the packets from the kernel.

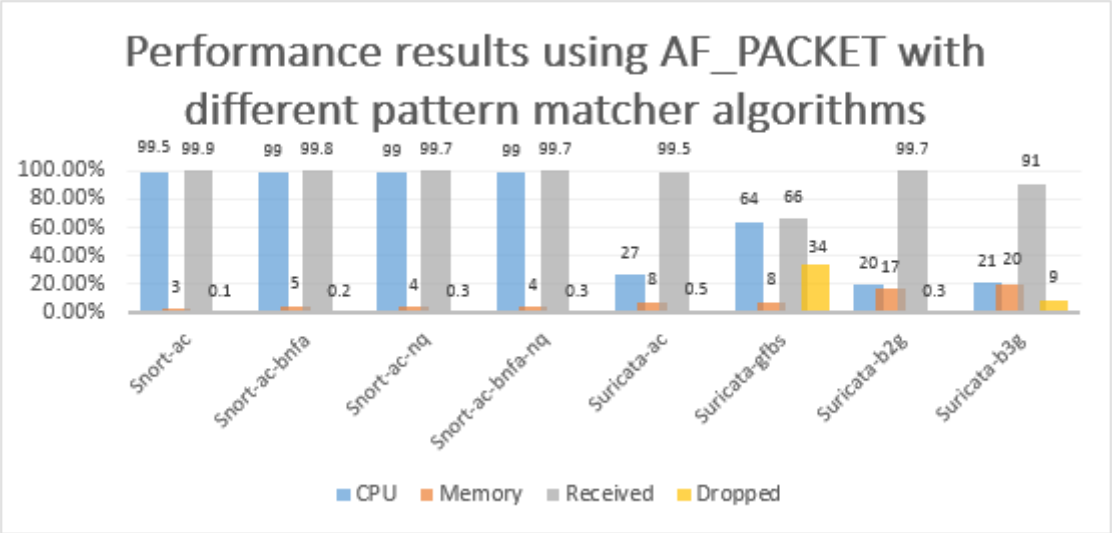

Figure 6.7: Percentage of performance utilized by Snort and Suricata after optimizations

Bro does not support AFPACKET capturing mode, so we only built test cases for Snort and Suricata for comparing the results to a previous libpacp results. In this experiment, we changed the data capture method to AFPACKRT, and evaluated all pattern matcher algorithms provided by Snort and Suricata. Figure 6.7 illustrates the performance results for Snort and Suricata when used on our 10 Gb/s campus network link.

Both IDSs have significantly improved results compared to the default DAQ (libpcap socket). With AFPACKET, most pattern matching algorithms were dropping less than 0.3% packets overall, however, the 'gfbs' and 'b3g' algorithms in Suricata did not manage to achieve the best results using AFPACKET, the former dropped 34% packets and the latter dropped 9%. The CPU usage remained similar to the previous results; Snort's CPU usage is affected by the number of mix flows. The average CPU usage for Suricata is 20% except the 'gfbs' algorithm (64%), we performed the 'gfbs' algorithm in UoA campus network link more than 10 times, but the results remained the same.

For a single flow or twenty flows, the results remained similar to the UoA testing results. Thus, there was no need to report those results here.

We also evaluated the PFRING capture method for Snort, Bro and Suricata. As already mentioned in the previous chapter, PFRING provides a circular (ring) buffer and the applications read packets from this buffer. It accelerates the packet capture operation by distributing packets to multiple application processes simultaneously. More detailed explanations can be found from the PFRING project homepage [134]. Suricata was able to use PFRING and generate similar performance results to AFPACKET. Unfortunately, we were not able to get PFRING working as a DAQ for Snort and Bro. For some reason DAQ would not load the PFRING capture library. We have sent the question to the Snort and Bro development team, we haven't got any response yet. But, based on the previous studies [135, 136], Using PFRING resulted in best performance from both IDSs based on the CPU usage and the packet drop rate.

- Experiment 4 - Optimize each IDS configuration by modifying the detection algorithms

Based on our experience, deploying a suitable IDS configuration for any network requires a lot of research and repeated evaluations. We have disabled some IDS detection algorithms based on the observed flow type. The results show a slight improvement after disabling the unused pre-processors or signatures, for example the drop rate for Bro decreased 30% after we disabled some irrelevant event handlers and polices. The memory usage remained low (28%), it was just the opposite for CPU usage (99%). Furthermore, it is important to mention that a misconfigured IDS could lead to disastrous security issues.

6.1.5 Discussion of experiment results

We summarise and analysis the results from all experiments in this section. All the test cases were performed more than 10 times in order to get an average result.

- Dropped Packets. In terms of dropped packets, Suricata and Snort achieved the desired result after optimizing the configuration and the capture method, 0.3% dropped packets were detected for our campus network link. Unfortunately some of Suricata's pattern matcher algorithms still dropped packets in UoA campus network link. For instance, the 'gfbs' pattern matcher algorithm in Suricata dropped 34% incoming traffic.
- CPU Usage. When we discuss the CPU usage, lower results are better. It seems that in the mixed flow environment, the single thread IDS generated a high CPU load on a single core processor, while Suricata reduced the load on a single CPU by utilizing all the processor cores. The average CPU usage for Suricata is 23%.
- Memory Usage. IDSs seem to use only a small amount of memory resources. Note that our different network environments did not have any significant effect on memory consumption. The memory for three IDSs is affected by the different searching algorithms, the DAQ method and the number of detection mechanisms loaded.

6.2 A new DNS reconnaissance detection solution

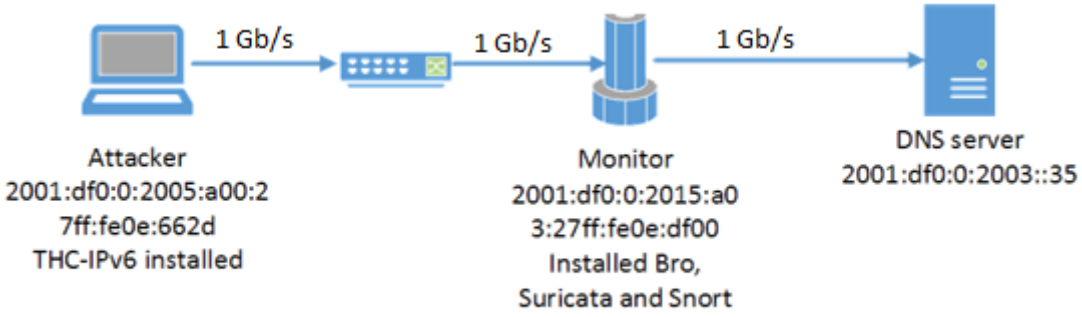

Figure 6.8: Logical network diagram for simulating DNS reconnaissance attack

In this section, we aim to understand whether is possible to use the existing solutions to detect the IPv6 DNS reconnaissance attack (see Section 3). We set up two environments to test the feasibility of this: a small experiment network (Figure 6.8) and the UoA network link (Figure 6.2). The results show that no IDS is able to capture the DNS reconnaissance attack using its default rules. In order to detect this new type of reconnaissance attack in IPv6 network, we propose a possible security solution. Figure 6.9 demonstrates the flow diagram of this new solution. Our detection algorithm focuses on the packet payload. When the traffic arrives at the host which has installed the IDS, the IDS picks up the traffic and matches it against the packet's payload in previous packets. If the flow hits the pre-configured condition, the alarm is generated. In our detection algorithm, each IDS checks the incoming packet content. If the packet contains a reverse lookup request, the IDS inserts the packet into a global data structure 'Flowtable' it then compares the request with the previous records for the same flow. If the IDS finds that the first nibble

of the current request is sequentially increased from previous requests, it increases the global value 'count' by one; when the count value for each single flow equals 15, the IDS generates an alert. On the other hand, if the incoming packet is not a reverse lookup request, this new mechanism won't be triggered and the packet will processed by other scripts. It is important to note that the pre-configured 'count' value can be modified by the users.

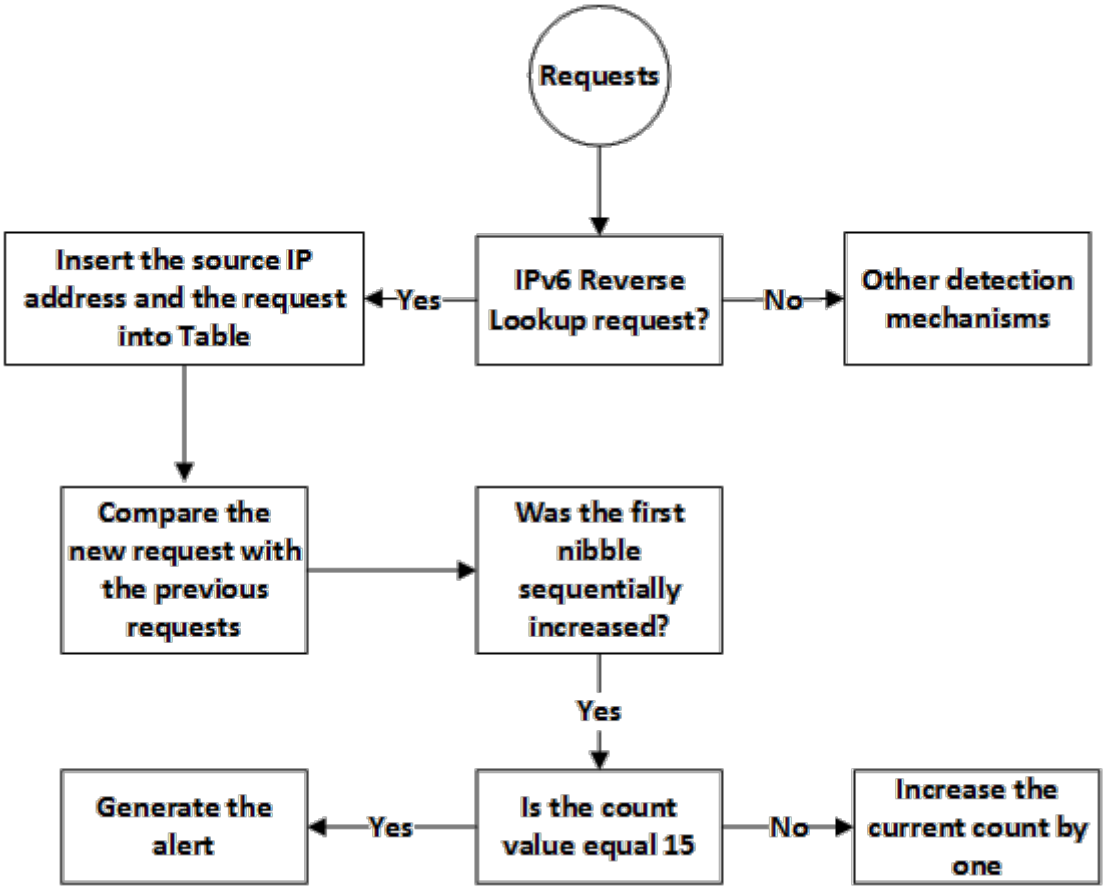

Figure 6.9: the flow diagram of detecting DNS reconnaissance attack

6.3 How feasible is it to implement the proposed solution in IDSs?

In the previous chapter we discussed the distinct differences between Snort and other two IDSs. Snort is a traditional signature-based IDS, it allows users to add a new rule for accessing each packet. However, there is a design limitation which prevents users from adding a data structure for saving the flow information or comparing the packet payload information with previous records that belong to the same flow. In contrast, Bro and Suricata both support script detection, their design allows users to examine network content and record the content from previous traffic. Therefore, we implemented our new mechanism in Bro and Suricata. Unfortunately, Snort does not have the functionality to save the payload information, so we were unable to implement this mechanism in Snort.

All in all, we find that there are only slight differences when implementing this new

mechanism in both IDSs. In the earlier version of Suricata, some API calls are not available, such as the flow information extracting function 'SCFlowTuple'; this was not introduced until Suricata 2.1. We also found only a few examples that show how to create a Suricata script and few documents that discuss the API calls at the earlier stage. In contrast, Bro gives more sample scripts and libraries for extracting packet information. Generally, its online documentation is well organized and maintained.

6.4 Evaluation of the proposed solution

We tested Bro and Suricata in both our test environment and in the real IPv6 campus network. When we launched the DNS reverse reconnaissance attack, both scripts detected the attack and generated the correct alarms. Here, there is a small challenge to use the same output format for both IDSs. In Suricata, users do not need to mention the output variable type; but Bro requires users to declare the output variable type. Figure 10 shows some alarm examples that were generated by Bro and Suricata, both IDSs were deployed on our campus network link. We have observed the attack from our test PC, but we have not yet observed this attack from outside to our campus DNS server. We assume that attackers may pay less attention to New Zealand IPv6 networks, because IPv6 is not widely deployed here, only 7% IPv6 adoption has been observed.[1]

Source IP	Port	Destination IP	Port	Prot	Query
2001:0df0:0000:2005:0a00:27ff:fe0e:662d	49290	2001:0df0:0000:2003:0000:0000:0000:0035	53	17	0.0.0.0.0.0.f.d.0.1.0.0.2.ip6.arpa
Source IP	Port	Destination IP	Port	Prot	Query
2001:0df0:0000:2005:0a00:27ff:fe0e:662d	49290	2001:0df0:0000:2003:0000:0000:0000:0035	53	17	1.0.0.0.0.0.f.d.0.1.0.0.2.ip6.arpa
Source IP	Port	Destination IP	Port	Prot	Query
2001:0df0:0000:2005:0a00:27ff:fe0e:662d	49290	2001:0df0:0000:2003:0000:0000:0000:0035	53	17	2.0.0.0.0.0.f.d.0.1.0.0.2.ip6.arpa
Source IP	Port	Destination IP	Port	Prot	Query
2001:0df0:0000:2005:0a00:27ff:fe0e:662d	49290	2001:0df0:0000:2003:0000:0000:0000:0035	53	17	3.0.0.0.0.0.f.d.0.1.0.0.2.ip6.arpa
Source IP	Port	Destination IP	Port	Prot	Query
2001:0df0:0000:2005:0a00:27ff:fe0e:662d	49290	2001:0df0:0000:2003:0000:0000:0000:0035	53	17	4.0.0.0.0.0.f.d.0.1.0.0.2.ip6.arpa
Source IP	Port	Destination IP	Port	Prot	Query
2001:0df0:0000:2005:0a00:27ff:fe0e:662d	49290	2001:0df0:0000:2003:0000:0000:0000:0035	53	17	5.0.0.0.0.0.f.d.0.1.0.0.2.ip6.arpa
Source IP	Port	Destination IP	Port	Prot	Query
2001:0df0:0000:2005:0a00:27ff:fe0e:662d	49290	2001:0df0:0000:2003:0000:0000:0000:0035	53	17	6.0.0.0.0.0.f.d.0.1.0.0.2.ip6.arpa
Source IP	Port	Destination IP	Port	Prot	Query
2001:0df0:0000:2005:0a00:27ff:fe0e:662d	49290	2001:0df0:0000:2003:0000:0000:0000:0035	53	17	7.0.0.0.0.0.f.d.0.1.0.0.2.ip6.arpa
Source IP	Port	Destination IP	Port	Prot	Query
2001:0df0:0000:2005:0a00:27ff:fe0e:662d	49290	2001:0df0:0000:2003:0000:0000:0000:0035	53	17	8.0.0.0.0.0.f.d.0.1.0.0.2.ip6.arpa
Source IP	Port	Destination IP	Port	Prot	Query
2001:0df0:0000:2005:0a00:27ff:fe0e:662d	49290	2001:0df0:0000:2003:0000:0000:0000:0035	53	17	9.0.0.0.0.0.f.d.0.1.0.0.2.ip6.arpa
Source IP	Port	Destination IP	Port	Prot	Query
2001:0df0:0000:2005:0a00:27ff:fe0e:662d	49290	2001:0df0:0000:2003:0000:0000:0000:0035	53	17	a.0.0.0.0.0.f.d.0.1.0.0.2.ip6.arpa
Source IP	Port	Destination IP	Port	Prot	Query
2001:0df0:0000:2005:0a00:27ff:fe0e:662d	49290	2001:0df0:0000:2003:0000:0000:0000:0035	53	17	b.0.0.0.0.0.f.d.0.1.0.0.2.ip6.arpa
Source IP	Port	Destination IP	Port	Prot	Query
2001:0df0:0000:2005:0a00:27ff:fe0e:662d	49290	2001:0df0:0000:2003:0000:0000:0000:0035	53	17	c.0.0.0.0.0.f.d.0.1.0.0.2.ip6.arpa
Source IP	Port	Destination IP	Port	Prot	Query
2001:0df0:0000:2005:0a00:27ff:fe0e:662d	49290	2001:0df0:0000:2003:0000:0000:0000:0035	53	17	d.0.0.0.0.0.f.d.0.1.0.0.2.ip6.arpa
Source IP	Port	Destination IP	Port	Prot	Query
2001:0df0:0000:2005:0a00:27ff:fe0e:662d	49290	2001:0df0:0000:2003:0000:0000:0000:0035	53	17	e.0.0.0.0.0.f.d.0.1.0.0.2.ip6.arpa
Source IP	Port	Destination IP	Port	Prot	Query
2001:0df0:0000:2005:0a00:27ff:fe0e:662d	49290	2001:0df0:0000:2003:0000:0000:0000:0035	53	17	f.0.0.0.0.0.f.d.0.1.0.0.2.ip6.arpa

Detecting the DNS reverse searching from:2001:0df0:0000:2005:0a00:27ff:fe0e:662d to 2001:0df0:0000:2003:0000:0000:0000:0035

Figure 6.10: Example of DNS reconnaissance attack logs for both Bro and Suricata

6.5 Conclusion

In this chapter, an overview of three popular open-source IDS was provided along with their comparative performance benchmarks. We evaluated the feasibility of using each of three IDSs in a high speed network.

[1] http://www.ipv6matrix.org/

First, we demonstrated the challenges of using IDSs in high speed networks, we proposed possible performance improvement by tuning the following factors: the pattern matcher algorithms, the packet capture methods and the network environment. In our network environment, Suricata with its multi-threaded architecture works better than Snort and Bro. However, it is important to note that different configurations can be applied for different environments. Our experiments show different performance results when the environment is changed. In order to reuse the existing Snort or Bro solutions, we found some alternative configurations for improving the performance of Snort and Bro. Abaid et al. [137], for example, introduced the use of an SDN splitting incoming flows and distributing traffic load across a range of Bro instances. In such system, each Bro instance only processes a small amount of traffic. We evaluated the different algorithms and methods for Snort, and observed a significant performance improvement after optimizing Snort's configuration in the high speed network.

Second, we found that the three IDSs cannot detect the DNS reconnaissance attack using their default configurations. By carefully investigating the detection mechanisms in the default rules, we find that all the detection mechanisms have good support for detecting the most common attacks, but there is no support for new IPv6 attacks. We proposed a new solution for detecting the DNS reconnaissance attack, we implemented this new solution in Bro and Suricata. As a result, we are able to detect the DNS reconnaissance attack from both inside and outside of our campus network. However, Snort does not allow us to implement this new mechanism. After analysing the Snort design structure, we found that Snort doesn's provide a global data structure for storing the flow information and cannot be modified.

All in all, we suggest that future IDSs should be designed to achieve the following requirements:

- Multi-threaded architecture: the single thread IDS used a larger amount of CPU resources, while a multi-threaded IDS distributes the operations to each core and uses less CPU capacity from each processor
- High speed packet capture method: Three packet methods have been evaluated in this chapter, it clearly shown that the AFPACK or PFRing is better than the default 'libpcap' based on the performance results from the high speed network. However, there is a pre-condition for using PFRing and AFPACK, the former needs the PFRing driver for the packet capture cards, while the latter only works in Linux environment. We need more general and fast packet capture methods in the high speed network environment
- IPv6 detection mechanism: we find little support for detecting new IPv6 attacks from the three IDSs, we suggest that IDS developers should put more IPv6 detection mechanisms in future releases
- Flow caching: A flow caching feature is important for analyzing malicious behaviors, future IDSs should include a good data structure for caching flow information

In the next chapter, we highlight our main contribution of this study, and answer our four research questions from Chapter 1. First we identify that network reconnaissance is one of the important indicators of an impending network attack, and explain the IPv6 DNS reconnaissance attack. Second, we discuss the technical challenges of using IDSs in high speed networks and IPv6 attack detections. Finally, we mention several avenues of future work.

Chapter 7

Conclusion

This chapter summarises the main findings of this thesis, including the changing association between IPv4 and IPv6 attacks, new attacks in IPv6, the configuration settings for using IDSs in high speed networks and the challenges of using existing IDSs to detect IPv6 attacks. In addition, we list some suggestions for further research following on from our project. The remainder of this chapter is organised as follows. Section 7.1 summarises the findings from the experiments and answers to our research questions. In addition, it discusses the main contribution of this study Section 7.2 discusses future work.

7.1 Research Conclusion

Q1. What kind of new attacks may be seen in IPv6?

In this study, we reviewed and analysed the existing IPv4 and new IPv6 security issues. We evaluated the feasibilities of launching existing IPv4 attacks in IPv6. We highlighted that the IPv6 protocol reduces some existing network layer issues and eliminates others, because some IPv4 features do not apply in IPv6. In addition, based on the experimental results, we observed that some current IPv4 penetration testing tools do not support IPv6 penetration testing while others only provide some basic features. In addition, we discussed some new attacks in IPv6, for instance attacks against new protocols: Neighbour Discovery (ND) spoofing attack and Duplicate Address Detection (DAD) DoS attack or attacks against the routers: Router Advertisement spoofing attack, RH0 (Type 0 Routing Headers) attack and ND cache exhaustion; we recommended that some IPv6 penetration testing tools should test this. By comparing the existing attacks and new attacks, we notice that most attacks share some common strategies, in particular a reconnaissance attack is an initial step toward finding hosts or system vulnerabilities. Spoofing attacks have been detected in both IPv4 and IPv6 as well as DoS attacks. However, the main difference is the way of bringing down a victim, for example, in IPv6, Neighbour discovery spoofing has been used to inject a malicious neighbor advertisement instead of ARP spoofing. Again, network reconnaissance is one of the important indicators of an impending network attack, many studies [24, 28] mentioned that the traditional reconnaissance attack is less feasible in IPv6 networks. Therefore, in our second research question, we

try to understand how will reconnaissance attacks change in IPv6 networks?

Q2. How will reconnaissance attacks change in IPv6 networks?

Because of IPv6's greatly increased address space, it was at first thought that it was infeasible to launch traditional address scanning attacks in IPv6, but some alternative methods have been proposed in the current IPv6 network. Such methods are based on using publicly available data to find IPv6 host addresses. We designed two surveys to investigate the feasibility of launching reconnaissance attacks in IPv6. In the first survey, we modified the DNS reconnaissance tool from the THC-IPv6 package to achieve our survey requirement. We launched our survey in five regions and fifty countries. The results show some potential issues in current DNS reverse zone deployment. For example, some network administrators save the IPv6 client addresses in the DNS reverse zone without any protection. This will lead to a security issue in that attackers can probe the DNS reverse zone to obtain the IPv6 client information. Again, if the IPv6 host addresses are using some predictable patterns, it will help the attackers to reduce the time needed for finding other hosts in the same network. Based on our findings, we described a few possible guides for users against IPv6 DNS reconnaissance attacks. For instance, In general the DNS reverse zone saves records for each individual IPv6 addresses. However, Durand et al. [138] discussed how to deploy wildcard reverse DNS records, for example "by configuring one name for a subnet (/64) or a site (/48). As a concrete example, a site (or the site's ISP) could configure the reverses of the prefix 2001:db8:f00::/48 to point to one name using a wildcard record like *.0.0.f.0.8.b.d.0.1.0.0.2.ip6.arpa. IN PTR site.example.com." [138], therefore the DNS reconnaissance searching will fail to gather the IPv6 address for each individual host in this prefix. We also introduced a new detection mechanism for use in Bro and Suricata; to help users detect DNS reconnaissance attacks.

In our second survey, we monitored IPv6 traffic from the UoA campus network for three months; the results demonstrate the trend of address allocation mechanisms usage around the world during the last five years (2009-2014). We found that some network administrators still do not allocate non-predictable values to IPv6 interface ID fields, but are using predictable patterns in their IIDs. Many previous studies already claimed that common patterns can be leveraged by attackers to cut down the search space for the IID field. Our results show that a high percentage of server addresses do not use a random IID allocation mechanism; we presume that this is because network administrators want to identify the servers as soon as possible if any security issues occur. In contrast, large numbers of IPv6 client addresses are generated by using a random IID allocation mechanism, which indicates that network administrators have started to consider privacy and security when they allocate an IID field for IPv6 clients. Again, we described a few techniques to use against IPv6 address scanning, such as the replacement of EUI-64 IID with privacy-based or randomized IID allocation mechanisms. Furthermore, we suggested that network administrators should avoid having predictable values in their IID fields.

Q3 What are the key architectural considerations for open source IDSs as

the network speeds the operate on increase?

In Chapter 6, we discussed factors that can affect an IDS's performance results, such as its pattern matcher algorithms, packet capture methods and the mix of background flows. We briefly discussed three popular open-source intrusion detection systems (IDS) along with their performance results in both experimental and our campus network link environments. We performed the evaluation on a 10 Gbit/s network with a number of test cases. The experiment results demonstrated that the default configurations were able to handle 10 Gb/s high network traffic with a small mix of flows, however the number of dropped packets increased significantly when we launched the same test in our campus network link. We applied various optimization techniques and tested different searching and DAQ methods. As a result, all the IDSs showed a significant improvement. Suricata with its multi-thread architecture achieved the best performance results in most test cases. Suricata only dropped 0.1% packets at our campus network when using the AFPACKET or the PFRING DAQ method paired with the 'ac' pattern matcher algorithm. From the information gathered from the previous chapters, we made suggestions of how future IDSs should be designed to compatible with high speed networks. These are:

- Multi-threading. As a multi-threaded design, IDSs can certainly offer increased speed and efficiency of CPU usage. The multi-threaded engine distributes the increased processing power to multi-core CPU chip sets and optimizes the CPU usage for each core.
- Packet capture method. For generic packet capture methods, we demonstrated the limitation of using AFPACKET and PFRING, the former only used in the Linux environment, the latter requires a network interface card that PFRING supports. New IDSs releasing should consider a new packet capture method that can handle the increased network speed, while being compatible with existing hardware and OS environments.
- Access encrypted traffic. HTTP 2.0 encrypts all the traffic by default to safeguard the integrity and confidentiality of data. However, encryption hides the majority of traffic content from any intrusion detection system monitoring traffic in the network. The network or system has become less secure due to the lack of IDS monitoring. In future releases, we hope that IDS developers add more features to handle encrypted traffic without changing existing network architectures. (such as the changes we mentioned in Chapter 4)
- User-friendly design. Tuning the IDS can be a long and arduous undertaking that requires both time and expertise, we want to make IDS as easy as possible for people to learn, configure and run. Developers need to put more documentation and examples in the configuration files, as well as using a GUI for the initial configuration.

Q4 How should an open source IDS be designed to more readily facilitate new emerging IPv6 attacks?

We evaluated and analysed the existing detection mechanisms in three popular open source IDSs (Snort, Bro and Suricata). The results show that the three IDSs have good support for detecting the most common attacks in IPv6 environment, such as port scanning, TCP SYN scanning, etc. However, none of them detect the IPv6 DNS reconnaissance attack or new IPv6 attacks. We designed and proposed a new solution for detecting the IPv6 DNS reconnaissance attack. We demonstrated the feasibility, or otherwise, of implementing this new mechanism in the three IDSs. Bro provides sample codes and references showing how to write a Bro script. In addition, Bro provides a well-designed data structure for saving and maintaining the flow information. Similar to Bro, Suricata also supports the flow information caching feature. However, scripting is initially more difficult for Suricata because of its lack of available documentation. Snort is not suitable for implementing this new mechanism because it does not provide a data structure for storing the flow information. We suggest Snort developers should create more flexible data structure and detection procedures for comparing the current packet with previous records. Again, there is only minimal IPv6 detection in the current IDSs, we suggest IDS developers should release more IPv6 rules or policies should handle emerging IPv6 threats.

7.2 Future Research

There are a number of areas for future research related to our study. We propose possible future work in this section.

- How feasible is it to launch other IPv6 attacks in real IPv6 environments?

We have mentioned a few new IPv6 attacks in Chapter 2. Since IPv6 introduced some new features, these changes may impact on the existing network security solutions. We have demonstrated some differences that affect both attacking and defending in Chapter 2, such as that some existing penetration tools do not support IPv6, and some IDSs do not detect the new IPv6 attacks. We have presented a possible IPv6 reconnaissance attack in the real IPv6 networks from both offense and defence perspectives. However, there are still more attacks that can be used to cause serious security issues in IPv6 networks. We plan to evaluate these new IPv6 vulnerabilities in the real IPv6 network and demonstrate the feasibility of various responses.

- How best to allocate IPv6 addresses in an IPv6 mobile devices?

In Chapter 3, we have discussed the issues pointed out by earlier studies [28, 24], for example some IID allocation mechanisms can generate security and privacy issues. For instance, EUI-64 exposes the user's activities. Therefore, we proposed some possible suggestion to minimise these issues. For instance, an IPv6 host should receive a randomized IPv6 address from a DHCPv6 server. However, using DHCPv6 seems less feasible for some mobile devices. Some earlier works [139, 140, 141] claim that the Android operating

system does not support DHCPv6, instead mobile devices can either obtain the IPv6 addresses through the Stateless Autoconfiguration (SLAAC) or DHCP-Prefix delegation (PD). If the mobile device is configured to use the former one that will certainly generate privacy issues. Through a comprehensive study of IPv6 address allocation mechanisms for the IPv6 mobile devices based on the privacy and security, we could contribute more to the knowledge base of IPv6 deployment.

- How to take advantage of signature and anomaly IDS approaches as a hybrid solution?

In Chapter 4, we have discussed the strengths and weaknesses of using both signature and anomaly detection. A signature detection solution requires searching for a series of bytes or packet payloads for network traffic. The main advantage of using this solution is that it is easy to install and configure, because it comes with a set of pre-defined rules and scripts. In addition, it shows a high level of accuracy in the alarms it generates, if the user knows the malicious behaviour being monitored. The signature based solution also has a few disadvantages, for instance it can't detect novel attacks, and users have to create a signature for each attack. Therefore, there is an arms race between attackers and IDS signature developers. In contrast, for the anomaly solution, a network administrator defines a description of accepted network behaviour instead of creating a signature for each attack. One benefit of doing this is that once a behaviour has been defined, the detection engine can scale more quickly and easily than the signature based mode, because there is little additional operation cost for creating a new signature for every attack and potential variant. However, the weakness of using anomaly detection solutions is their high false alarm rates. We see how the strengths of one detection method counterbalance the weaknesses of another. It would be very useful if we can build a hybrid solution to make both solutions complement each another.

Bibliography

[1] D. Plonka and A. Berger, "Temporal and Spatial Classification of Active IPv6 Addresses," *arXiv preprint arXiv:1506.08134*, 2015.

[2] Q. Hu and N. Brownlee, "IPv6 Host Address Usage Survey," *International Journal of Future Computer and Communication*, vol. 3, no. 5, p. 341, 2014.

[3] N. Brownlee and H. Qinwen, "How Interface ID Allocation Mechanisms are Performed in IPv6," in *Proceedings of the 2014 CoNEXT on Student Workshop*. ACM, 2014, pp. 26–27.

[4] Consultancy.uk. (2016) Integrated cyber approach key for manufacturing. Consultancy.uk. Date viewed: 2016. [Online]. Available: http://www.consultancy.uk/news/1665/integrated-cyber-approach-key-for-manufacturing

[5] A. Q. S. S. Report. (2016) Record Number Of DDoS Attacks Recorded On Akamai Routed Network In Q3 2015. Akamai Q3 SOTI Security Report. Date viewed: 2016. [Online]. Available: http://www.econotimes.com/Record-Number-Of-DDoS-Attacks-Recorded-On-Akamai-Routed-Network-In-Q3-2015-%E2%80%93Report-142490

[6] BUSINESS VALUE OF SECURITY AND CONTROL. Date viewed: 2016. [Online]. Available: http://paginas.fe.up.pt/ acbrito/laudon/ch10/chpt10-2main.htm

[7] Google. (2016) Google IPv6 Statistics. Google. Date viewed: 2016. [Online]. Available: https://www.google.com/intl/en/ipv6/statistics.html

[8] S. Kumar. (2016) Survey of current network intrusion detection techniques. Date viewed: 2016. [Online]. Available: http://www.cse.wustl.edu/ jain/cse571-07/ftp/ids.pdf

[9] Libpcap. (2016) LibPcap. Date viewed: 2016. [Online]. Available: http://www.programming-pcap.aldabaknocking.com/extrastuff.htmll

[10] R. Larsen. (2016) Protocol Detection Capabilities in Bro. Date viewed: 2016. [Online]. Available: https://digirola.files.wordpress.com/2012/11/imt4022-digitalforensicsii_rogerlarsen_2012-06_final.pdf

[11] M. CORKERY. (2016) Once Again, Thieves Enter Swift Financial Network and Steal. The New York Times. Date viewed: 2016. [Online]. Available: http://www.nytimes.com/2016/05/13/business/dealbook/swift-global-bank-network-attack.html

[12] Chuang. (2016) Denial of Service Attacks. Utexas. Date viewed: 2016. [Online]. Available: http://www.cs.utexas.edu/users/chuang/dos.html

[13] P. Muncaster. (2016) China Launches Man in the Middle Attack Against Google. Info Security. Date viewed: 2016. [Online]. Available: http://www.infosecurity-magazine.com/news/china-man-in-the-middle-attack/

[14] S. Farrell and H. Tschofenig, "Pervasive Monitoring Is an Attack," *IETF RFC 7258*, 2014.

[15] N. staff. (2016) Major security breach reveals personal details online. Acurrent Affair. Date viewed: 2016. [Online]. Available: http://aca.nine.com.au/article/8833611/major-security-breach-reveals-personal-details-online

[16] McAfee. (2016) McAfee Labs Threats Report. McAfee. Date viewed: 2016. [Online]. Available: http://www.mcafee.com/hk/threat-center.aspx

[17] S. Center. (2016) Internet Security Threat Report 2016. Symantec. Date viewed: 2016. [Online]. Available: https://www.symantec.com/security-center/threat-report

[18] J. Vijayan. (2016) 4 Worst Government Data Breaches Of 2014. InformationWeek. Date viewed: 2016. [Online]. Available: http://www.informationweek.com/government/cybersecurity/4-worst-government-data-breaches-of-2014/d/d-id/1318061

[19] M. Shuff. (2016) Learning from TalkTalk: what every CEO needs to know. BBC News. Date viewed: 2016. [Online]. Available: https://www.linkedin.com/pulse/learning-from-talktalk-what-every-ceo-needs-know-michael-shuff

[20] C. Riley. (2016) Hackers stole millions in third attack on global banking system. CNN Money. Date viewed: 2016. [Online]. Available: http://money.cnn.com/2016/05/20/news/swift-bank-attack-global-ecuador/

[21] W. Lee and B. Rotoloni. (2016) Emerging cyber threats report for 2015. Georgia Tech Cyber Security Summit 2014. Date viewed: 2016. [Online]. Available: https://www.gtisc.gatech.edu/pdf/Threats_Report_2015.pdf

[22] A. Networks. (2016) Arbor Networks Reports Unprecedented Spike in DDoS Attack Size Driven by NTP Misuse. Arbor Networks. Date viewed: 2016. [Online]. Available: https://www.arbornetworks.com/arbor-networks-reports-unprecedented-spike-in-ddos-attack-size-driven-by-ntp-misuse

[23] J. Titcomb. (2016) Android security breach puts millions at risk of smartphone hijacking. The Telegrapgh. Date viewed: 2016. [Online]. Available: http://www.telegraph.co.uk/technology/internet-security/11788184/Android-security-breach-puts-millions-at-risk-of-smartphone-hijacking.html

[24] T. Chown, "IPv6 implications for network scanning," *IETF RFC 5157*, 2008.

[25] B. Carpenter and S.Jiang, "Transmission and Processing of IPv6 Extension Headers," *IETF RFC 7045*, 2013.

[26] P. van Dijk. (2016) Finding v6 hosts by e ciently mapping ip6.arpa. 7bits. Date viewed: 2016. [Online]. Available: http://7bits.nl/blog/posts/finding-v6-hosts-by-efficiently-mapping-ip6-arpa

[27] G. V. de Velde, T.Hain, R.Droms, B.Carpenter, and E.Klein, "Local Network Protection for IPv6," *IETF RFC 4864*, 2007.

[28] T. Chown and F. Gont, "Network reconnaissance in IPv6 networks," *IETF RFC 7707*, 2016.

[29] J. I.Gashinsky and W.Kumari, "Operational Neighbor Discovery Problems," *IETF RFC 6583*, 2012.

[30] Y. C. Bao, X.Li and W. Shang, "dIVI: Dual-Stateless IPv4/IPv6 Translation," *IETF draft-xli-behave-divi-06*, 2014.

[31] S. A.Durand and J.Paugh, "Issues with Dual Stack IPv6 on by Default," *IETF draft-ietf-v6ops-v6onbydefault-03.txt*, 2004.

[32] D. Minoli and J. Kouns, *Security in an ipv6 environment.* CRC Press, 2016.

[33] J. Czyz, M. Luckie, M. Allman, and M. Bailey, "Dont Forget to Lock the Back Door A Characterization of IPv6 Network Security Policy," in *Proceedings of the 23rd Annual Network and Distributed System Security Symposium, San Diego, California*, vol. 389, 2016.

[34] F.Gont, "Processing of IPv6 "Atomic" Fragments," *IETF RFC 6946*, 2013.

[35] S.Krishnan, "Handling of Overlapping IPv6 Fragments," *IETF RFC 5722*, 2009.

[36] M. Kumar, B. K. Mishra, and T. Panda, "Predator-Prey Models on Interaction between Computer Worms, Trojan Horse and Antivirus Software Inside a Computer System," *International Journal of Security and its Applications*, vol. 10, no. 1, pp. 173–190, 2016.

[37] C. M. Weir, "Using probabilistic techniques to aid in password cracking attacks," 2016.

[38] E. Adi, Z. A. Baig, P. Hingston, and C.-P. Lam, "Distributed denial-of-service attacks against HTTP/2 services," *Cluster Computing*, vol. 19, no. 1, pp. 79–86, 2016.

[39] H. Ma, H. Ding, Y. Yang, Z. Mi, J. Y. Yang, and Z. Xiong, "Bayes-based ARP attack detection algorithm for cloud centers," *Tsinghua Science and Technology*, vol. 21, no. 1, pp. 17–28, 2016.

[40] H. Son and M. Riley. (2016) JPMorgan Password Leads Hackers to 76 Million Households. Bloomberg. Date viewed: 2016. [Online]. Available: http://www.bloomberg.com/news/articles/2014-10-03/jpmorgan-password-said-to-lead-hackers-to-76-million-households

[41] H. Williams. (2016) NERC Report Highlights Lessons Learned from Ukraine Electric Utility Cyber Attack. Hunton Williams. [Online]. Available: https://www.huntonprivacyblog.com/2016/03/31/nerc-report-highlights-lessons-learned-from-ukraine-electric-utility-cyber-attack/

[42] A. Gupta and S. K. Yadav, "SQL Injection Attack on Web Application: A Review," *Imperial Journal of Interdisciplinary Research*, vol. 2, no. 6, 2016.

[43] B. K. Mishra, "Mathematical Model on Attack of Worm and Virus in Computer Network," *International Journal of Future Generation Communication and Networking*, vol. 9, no. 6, pp. 245–254, 2016.

[44] S. Das, H. Xiao, Y. Liu, and W. Zhang, "Online malware defense using attack behavior model," in *2016 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2016, pp. 1322–1325.

[45] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Hypertext Transfer Protocol – HTTP/1.1," *IETF RFC 2616*, 1999.

[46] U. S. Qurashi and Z. Anwar, "AJAX based attacks: Exploiting Web 2.0," in *Emerging Technologies (ICET), 2012 International Conference on*. IEEE, 2012, pp. 1–6.

[47] M. Vijayalakshmi, S. M. Shalinie, and A. A. Pragash, "IP traceback system for network and application layer attacks," in *Recent Trends In Information Technology (ICRTIT), 2012 International Conference on.* IEEE, 2012, pp. 439–444.

[48] V. Durcekova, L. Schwartz, and N. Shahmehri, "Sophisticated denial of service attacks aimed at application layer," in *ELEKTRO, 2012.* IEEE, 2012, pp. 55–60.

[49] G. Dong, Y. Zhang, X. Wang, P. Wang, and L. Liu, "Detecting cross site scripting vulnerabilities introduced by HTML5," in *Computer Science and Software Engineering (JCSSE), 2014 11th International Joint Conference on.* IEEE, 2014, pp. 319–323.

[50] M. A. Prabakar, M. KarthiKeyan, and K. Marimuthu, "An efficient technique for preventing SQL injection attack using pattern matching algorithm," in *Emerging Trends in Computing, Communication and Nanotechnology (ICE-CCN), 2013 International Conference on.* IEEE, 2013, pp. 503–506.

[51] S. Gupta and B. Gupta, "XSS-SAFE: a server-side approach to detect and mitigate cross-site scripting (XSS) attacks in JavaScript code," *Arabian Journal for Science and Engineering*, vol. 41, no. 3, pp. 897–920, 2016.

[52] S. P. Singh and M. UpendraNathTripathi, "Detection and prevention of sql injection attack using hashing technique," *International Journal of Modern Communication Technologies & Research*, vol. 2, 2014.

[53] K. Alminshid and M. N. Omar, "Detecting backdoor using stepping stone detection approach," in *Informatics and Applications (ICIA), 2013 Second International Conference on.* IEEE, 2013, pp. 87–92.

[54] K. ZETTER. (2016) Secret Code Found in Juniper's Firewalls Shows Risk of Government Backdoors. Wired. Date viewed: 2016. [Online]. Available: https://www.wired.com/2015/12/juniper-networks-hidden-backdoors-show-the-risk-of-government-backdoors/

[55] M. H. Almeshekah, C. N. Gutierrez, M. J. Atallah, and E. H. Spafford, "Ersatzpasswords–ending password cracking," 2015.

[56] K. Alminshid and M. N. Omar, "Detecting backdoor using stepping stone detection approach," in *Informatics and Applications (ICIA), 2013 Second International Conference on.* IEEE, 2013, pp. 87–92.

[57] R. Swanink, E. Poll, and P. Schwabe, "Persistent effects of man-in-the-middle attacks," 2016.

[58] S. T. Zargar, J. Joshi, and D. Tipper, "A survey of defense mechanisms against distributed denial of service (DDoS) flooding attacks," *IEEE Communications Surveys & Tutorials*, vol. 15, no. 4, pp. 2046–2069, 2013.

[59] B. Biggio, Z. Akhtar, G. Fumera, G. L. Marcialis, and F. Roli, "Security evaluation of biometric authentication systems under real spoofing attacks," *IET biometrics*, vol. 1, no. 1, pp. 11–24, 2012.

[60] F. Najjar and H. El-Taj, "IPv6 Change Threats Behavior," *International Journal of Advanced Computer Science and Applications*, vol. 6, no. 1, 2015.

[61] Veritas. (2016) Security report indicates vulnerabilities on port 5634 used by VOM (Veritas Operations Manager). Veritas. Date viewed: 2016. [Online]. Available: https://www.veritas.com/support/en_US/article.000100551

[62] O. Al-Jarrah and A. Arafat, "Network Intrusion Detection System using attack behavior classification," in *Information and Communication Systems (ICICS), 2014 5th International Conference on.* IEEE, 2014, pp. 1–6.

[63] S. K. Patel and A. Sonker, "Rule-Based Network Intrusion Detection System for Port Scanning with Efficient Port Scan Detection Rules Using Snort," *International Journal of Future Generation Communication and Networking*, vol. 9, no. 6, pp. 339–350, 2016.

[64] Y. Medjadba and S. Sahraoui, "Intrusion Detection System to Overcome a Novel Form of Replay Attack (Data Replay) in Wireless Sensor Networks," *International Journal of Computer Network & Information Security*, vol. 8, no. 7, 2016.

[65] D. Plummer, "Ethernet Address Resolution Protocol: Or converting network protocol addresses to 48.bit Ethernet address for transmission on Ethernet hardware," *IETF RFC 826*, 1982.

[66] I. University. (2016) ARP Poison Routing an Attack at Indiana University. Indiana University. Date viewed: 2016. [Online]. Available: https://net.educause.edu/ir/library/powerpoint/SEC08078.pps

[67] D. Research. (2016) The New Threat: Targeted Internet Traffic Misdirection. DYN Research. Date viewed: 2016. [Online]. Available: http://research.dyn.com/2013/11/mitm-internet-hijacking/

[68] F. H. M. Ali, R. Yunos, and M. A. M. Alias, "Simple port knocking method: Against TCP replay attack and port scanning," in *Cyber Security, Cyber Warfare and Digital Forensic (CyberSec), 2012 International Conference on.* IEEE, 2012, pp. 247–252.

[69] F. D.McPherson and J. Halpern, "Source Address Validation Improvement (SAVI) Threat Scope," *IETF RFC 6959*, 2013.

[70] P. Limmaneewichid and W. Lilakiatsakun, "P-ARP: a novel enhanced authentication scheme for securing ARP," in *International Conference on Telecommunication Technology and Applications, Proceedings of CSIT*, 2011.

[71] W. Zhou, "Keynote: Detection of and Defense Against Distributed Denial-of-Service (DDoS) Attacks," in *2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications*. IEEE, 2012, pp. lxxxii–lxxxii.

[72] S. Haris, R. Ahmad, M. Ghani, and G. M. Waleed, "TCP SYN flood detection based on payload analysis," in *Research and Development (SCOReD), 2010 IEEE Student Conference on*. IEEE, 2010, pp. 149–153.

[73] M. Prince. (2016) The DDoS That Almost Broke the Internet. Cloudflare. Date viewed: 2016. [Online]. Available: https://blog.cloudflare.com/the-ddos-that-almost-broke-the-internet/

[74] M. Dunn. (2016) Service Incident. Zendesk. Date viewed: 2016. [Online]. Available: https://support.zendesk.com/hc/en-us/articles/218654547-Service-Incident-April-12th-2016

[75] A. S. Jose and A. Binu, "Automatic Detection and Rectification of DNS Reflection Amplification Attacks with Hadoop MapReduce and Chukwa," in *Advances in Computing and Communications (ICACC), 2014 Fourth International Conference on*. IEEE, 2014, pp. 195–198.

[76] L. Arshadi and A. H. Jahangir, "Entropy based SYN flooding detection," in *Local Computer Networks (LCN), 2011 IEEE 36th Conference on*. IEEE, 2011, pp. 139–142.

[77] J. Bi, B. Liu, J. Wu, and Y. Shen, "Preventing IP source address spoofing: A two-level, state machine-based method," *Tsinghua Science & Technology*, vol. 14, no. 4, pp. 413–422, 2009.

[78] Y. Wang and J. Chen, "Hijacking spoofing attack and defense strategy based on Internet TCP sessions," in *Instrumentation and Measurement, Sensor Network and Automation (IMSNA), 2013 2nd International Symposium on*. IEEE, 2013, pp. 507–509.

[79] A. Eden. (2016) Incident Report: NTP Attack or Why We Switched to Anycast Faster than Planned. dnsimple blog. Date viewed: 2016. [Online]. Available: https://blog.dnsimple.com/2014/02/post-mortem-or-why-we-switched-to-anycast-so-quickly/

[80] B. Blevins. (2016) SSDP DDoS attacks driving up average DDoS sizes. TechTarget. Date viewed: 2016. [Online]. Available: http://searchsecurity.techtarget.com/news/2240235194/SSDP-DDoS-attacks-driving-up-average-DDoS-sizes

[81] Arbor. (2016) Arbor Networks Detects Largest Ever DDoS Attack in Q1 2015 DDoS Report. Arbor. Date viewed: 2016. [Online]. Available: https://www.arbornetworks.com/arbor-networks-detects-largest-ever-ddos-attack-in-q1-2015-ddos-report

[82] M. Mimoso. (2016) Targeted Attack Uses Heartbleed to Hijack VPN Sessions. ThreatPost. Date viewed: 2016. [Online]. Available: https://threatpost.com/targeted-attack-uses-heartbleed-to-hijack-vpn-sessions/105567/

[83] P. Ferguson and D. Senie, "Network ingress filtering: Defeating Denial of Service (DoS) attacks which employ IP source address spoofing," *IETF RFC 2827*, 2000.

[84] A. Alabrah, J. Cashion, and M. Bassiouni, "Enhancing security of cookie-based sessions in mobile networks using sparse caching," *International journal of information security*, vol. 13, no. 4, pp. 355–366, 2014.

[85] T. Aura, "Cryptographically Generated Addresses (CGA)," *IETF RFC 3972*, 2005.

[86] J. J. Igor Gashinsky and W. Kumari, "Operational Neighbor Discovery Problems," *IETF RFC 6583*, 2012.

[87] J. Small. (2016) IPV6 ATTACKS AND COUNTERMEASURES. CDW Advanced Technology Services. Date viewed: 2016. [Online]. Available: http://www.rmv6tf.org/wp-content/uploads/2013/04/5-IPv6-Attacks-and-Countermeasures-v1.2.pdf

[88] P. Nikander, J. Kempf, and E. Nordmark, "IPv6 Neighbor Discovery (ND) trust models and threats," *IETF RFC 3756*, 2004.

[89] Y. C. Bao, X.Li and W. Shang, "IPv6 Addressing of IPv4/IPv6 Translators," *draft-xli-behave-divi-06*, 2014.

[90] T. Narten, S. Thomson, and T. Jinmei, "IPv6 Stateless Address Autoconfiguration," *IETF RFC 4862*, 2007.

[91] S. Sarma, "EVALUATION OF IPV6 SECURE NEIGHBOR AND ROUTER DISCOVERY PROTOCOL, USING LOCALLY AUTHENTICATION PROCESS," *Advance and Innovative Research*, vol. 2, no. 1, p. 14, 2015.

[92] B. Vrat, N. Aggarwal, and S. Venkatesan, "Anomaly Detection in IPv4 and IPv6 networks using machine learning," in *2015 Annual IEEE India Conference (INDICON)*. IEEE, 2015, pp. 1–6.

[93] C. P. E. Levy-Abegnoli, G. Van de Velde and J. Mohacsi, "IPv6 Router Advertisement Guard," *IETF RFC 7113*, 2011.

[94] A. H. NasserElDeen, "ICMPv6 Router Advertisement Flooding."

[95] F. Najjar, M. M. Kadhum, and H. El-Taj, "Detecting Neighbor Discovery Protocol-Based Flooding Attack Using Machine Learning Techniques," in *Advances in Machine Learning and Signal Processing*. Springer, 2016, pp. 129–139.

[96] R. M. Hinden and S. E. Deering, "IPv6 Multicast Address Assignments," *IETF RFC 2375*, 1998.

[97] S. E. Deering and R. M. Hinden, "IP version 6 addressing architecture," *IETF RFC 4291*, 2006.

[98] S. P. Naidu and A. Patcha, "IPv6: Threats Posed by Multicast Packets, Extension Headers and Their Counter Measures," *International Journal of Computer Science and Network Security (IJCSNS)*, vol. 14, no. 10, p. 71, 2014.

[99] R. Bonica, V. Manral, and F. Gont, "Implications of oversized IPv6 header chains," *IETF RFC 7112*, 2014.

[100] S. Krishnan, "Handling of Overlapping IPv6 Fragments," *IETF RFC 5722*, 2009.

[101] C. Bao, C. Huitema, M. Bagnulo, M. Boucadair, and X. Li, "IPv6 Addressing of IPv4/IPv6 Translators," *IETF RFC 6052*, 2010.

[102] E. Nordmark, "Stateless IP/ICMP Translation Algorithm (SIIT)," *IETF RFC 2765*, 2000.

[103] (2016) Regional internet registries statistics. Date viewed: 2016. [Online]. Available: http://www-public.it-sudparis.eu/maigron/RIR_Stats

[104] B. Carpenter, T. Chown, F. Gont, S. Jiang, A. Petrescu, and A. Yourtchenko, "Analysis of the 64-bit Boundary in IPv6 Addressing," *IETF RFC 7421*, 2015.

[105] M. O'Dell, "8+8-An alternate addressing architecture for IPv6," *IETF Internet draft draftodell-8+8-00*, 1996.

[106] RJ, Atkinson and SN, Bhatti, "Identifier-Locator Network Protocol (ILNP) Engineering Considerations," *IETF RFC 6741*, 2012.

[107] E. Karpilovsky, A. Gerber, D. Pei, J. Rexford, and A. Shaikh, "Quantifying the extent of IPv6 deployment," in *Passive and Active Network Measurement*. Springer, 2009, pp. 13–22.

[108] D. Malone, "Observations of IPv6 addresses," in *Passive and Active Network Measurement*. Springer, 2008, pp. 21–30.

[109] S. Thomson, C. Huitema, V. Ksinant, and M. Souissi, "DNS Extensions to support IP version 6," *IETF RFC 1886*, 1995.

[110] T. Narten, R. Draves, and S. Krishnan, "Privacy extensions for stateless address autoconfiguration in IPv6," *IETF RFC 4941*, 2007.

[111] F. Gont, "A Method for Generating Semantically Opaque Interface Identifiers with IPv6 Stateless Address Autoconfiguration (SLAAC," *IETF RFC 7217*, 2014.

[112] R.Droms, "Stateless Dynamic Host Configuration Protocol (DHCP) Service for IPv6," *IETF RFC 3736*, 2004.

[113] W. Shen, Y. Chen, Q. Zhang, Y. Chen, B. Deng, X. Li, and G. Lv, "Observations of IPv6 traffic," in *Computing, Communication, Control, and Management, 2009. CCCM 2009. ISECS International Colloquium on*, vol. 2. IEEE, 2009, pp. 278–282.

[114] S. R.M.Hinden and E. Nordmark, "IPv6 global unicast address format," *IETF RFC 3587*, 2003.

[115] L. Howard, "Reverse DNS in IPv6 for Internet Service Providers," *IETF draft-ietf-dnsop-isp-ip6rdns-01*, 2015.

[116] symantec. (2016) Internet Security Threat Report. symantec. Date viewed: 2016. [Online]. Available: https://www.symantec.com/content/dam/symantec/docs/reports/istr-21-2016-en.pdf

[117] A. V. Aho and M. J. Corasick, "Efficient string matching: an aid to bibliographic search," *Communications of the ACM*, vol. 18, no. 6, pp. 333–340, 1975.

[118] C.-C. Lo, Y.-T. Chen, and Y.-W. Chuang, "The network address translation problem of the home automation network: issues and solutions," in *TENCON'02. Proceedings. 2002 IEEE Region 10 Conference on Computers, Communications, Control and Power Engineering*, vol. 2. IEEE, 2002, pp. 720–723.

[119] J. Qu, G. Zhang, Z. Fang, and J. Liu, "A Parallel Algorithm of String Matching Based on Message Passing Interface for Multicore Processors," *International Journal of Hybrid Information Technology*, vol. 9, no. 3, pp. 31–38, 2016.

[120] J. E. Hopcroft, *Introduction to automata theory, languages, and computation.* Pearson Education India, 1979.

[121] Y. Gong, Q. Liu, X. Shao, C. Pan, and H. Jiao, "A novel regular expression matching algorithm based on multi-dimensional finite automata," in *2014 IEEE 15th International Conference on High Performance Switching and Routing (HPSR).* IEEE, 2014, pp. 90–97.

[122] R. Vidal, Y. Ma, and S. S. Sastry, "Principal Component Analysis," in *Generalized Principal Component Analysis.* Springer, 2016, pp. 25–62.

[123] L. Mechtri, F. D. Tolba, and N. Ghoualmi, "Intrusion detection using principal component analysis," in *Engineering Systems Management and Its Applications (ICESMA), 2010 Second International Conference on.* IEEE, 2010, pp. 1–6.

[124] C. T. Symons and J. M. Beaver, "Nonparametric semi-supervised learning for network intrusion detection: combining performance improvements with realistic in-situ training," in *Proceedings of the 5th ACM workshop on Security and artificial intelligence.* ACM, 2012, pp. 49–58.

[125] W. Tian and J. Liu, "A new network intrusion detection identification model research," in *Informatics in Control, Automation and Robotics (CAR), 2010 2nd International Asia Conference on*, vol. 2. IEEE, 2010, pp. 9–12.

[126] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: synthetic minority over-sampling technique," *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.

[127] M. Kumar, M. Hanumanthappa, and T. S. Kumar, "Encrypted Traffic and IPsec Challenges for Intrusion Detection System," in *Proceedings of International Conference on Advances in Computing.* Springer, 2013, pp. 721–727.

[128] S. McCanne and V. Jacobson, "The BSD Packet Filter: A New Architecture for User-level Packet Capture." in *USENIX winter*, vol. 46, 1993.

[129] F. Fusco and L. Deri, "High speed network traffic analysis with commodity multi-core systems," in *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement.* ACM, 2010, pp. 218–224.

[130] I. Bro. (2016) Bro Homepage. Date viewed: 2016. [Online]. Available: http://www.bro-ids.org

[131] Snort. (2016) PHF vulnerability. Date viewed: 2016. [Online]. Available: http://insecure.org/sploits/phf-cgi.html

[132] Y. Rekhter and T. Li, "An Architecture for IP Address Allocation with CIDR," *IETF RFC 1518*, 1993.

[133] S. Y. Yu, N. Brownlee, and A. Mahanti, "Performance and Fairness Issues in Big Data Transfers," in *Proceedings of the 2014 CoNEXT on Student Workshop*. ACM, 2014, pp. 9–11.

[134] ntop. (2016) PF_Ring. Date viewed: 2016. [Online]. Available: http://www.ntop.org/products/packet-capture/pf_ring/

[135] MetaFlows. (2016) Open Source IDS Multiprocessing With PF_RING. Date viewed: 2016. [Online]. Available: https://www.metaflows.com/features/pf_ring/

[136] SANS. (2016) Open Source IDS High Performance Shootout. Date viewed: 2016. [Online]. Available: https://www.sans.org/reading-room/whitepapers/intrusion/open-source-ids-high-performance-shootout-35772

[137] Z. Abaid, M. Rezvani, and S. Jha, "MalwareMonitor: An sdn-based framework for securing large networks," in *Proceedings of the 2014 CoNEXT on Student Workshop*. ACM, 2014, pp. 40–42.

[138] A. Durand, J. Ihren, and P. Savola, "Operational Considerations and Issues with IPv6 DNS," *IETF RFC 4472*, 2006.

[139] Soulskill. (2016) IT Pros Blast Google Over Android's Refusal To Play Nice With IPv6. Date viewed: 2016. [Online]. Available: https://tech.slashdot.org/story/15/06/23/2047259/it-pros-blast-google-over-androids-refusal-to-play-nice-with-ipv6

[140] (2016) Does Android have support for IPv6. Date viewed: 2016. [Online]. Available: http://android.stackexchange.com/questions/3718/does-android-have-support-for-ipv6

[141] J. Sanders. (2016) Android's lack of DHCPv6 support poses security and IPv6 deployment issues. Date viewed: 2016. [Online]. Available: http://www.techrepublic.com/article/androids-lack-of-dhcpv6-support-poses-security-and-ipv6-deployment-issues/