5-2016

Enhancing snort IDs performance using data mining

Mohammed Ali Almaleki

United Arab Emirates University

College of Information Technology

Information Security Track

ENHANCING SNORT IDS PERFORMANCE USING DATA MINING

Mohammed Ali Almaleki

This thesis is submitted in partial fulfillment of the requirements for the degree of

Master of Science in Information Security

Under the Supervision of Dr. Mohammad Mehedy Masud

May 2016

Declaration of Original Work

I, Mohammed Ali Almaleki, the undersigned, a graduate student at the United Arab Emirates University (UAEU), and the author of this thesis entitled **"***Enhancing Snort Ids Performance Using Data Mining"* hereby, solemnly declare that this thesis is my own original research work that has been done and prepared by me under the supervision of Dr. Mohammad Mehedy Masud in the College of Information Technology at UAEU. This work has not previously been presented or published, or formed the basis for the award of any academic degree, diploma or a similar title at this or any other university. Any materials borrowed from other sources (whether published or unpublished) and relied upon or included in my thesis have been properly cited and acknowledged in accordance with appropriate academic conventions. I further declare that there is no potential conflict of interest with respect to the research, data collection, authorship, presentation and/or publication of this thesis.

Student's Signature:_____ Date: _____

Approval of the Master Thesis

This Master Thesis is approved by the following Examining Committee Members:

1) Advisor (Committee Chair): Dr. Mohammad Mehedy Masud

   Title: Assistant Professor

   Information Security Track

   College of Information Technology

   Signature: _____ Date: 7/06/2016

2) Member: Dr. Zouheir Trabelsi

   Title: Associate Professor

   Information Security Track

   College of Information Technology

   Signature: _____ Date: 07/06/2016

3) Member (External Examiner): Dr. Amjad Gawanmeh

   Title: Assistant Professor

   Department of Computer and Software Engineering

   Institution: Khalifa University

   Signature: _____ Date: 19/06/2016

   Dr. Zouheir Trabelsi
   for Dr. Amjad Gawanmeh

This Master Thesis is accepted by:

Dean of the College of Information Technology: Prof. Omar El-Gayar

Signature _____ Date June 16, 2016

Dean of the College of the Graduate Studies: Professor Nagi T. Wakim

Signature _____ Date 19/6/2016

Copy 6 of 7

Abstract

Intrusion detection systems (IDSs) such as Snort apply deep packet inspection to detect intrusions. Usually, these are rule-based systems, where each incoming packet is matched with a set of rules. Each rule consists of two parts: the rule header and the rule options. The rule header is compared with the packet header. The rule options usually contain a signature string that is matched with packet content using an efficient string matching algorithm. The traditional approach to IDS packet inspection checks a packet against the detection rules by scanning from the first rule in the set and continuing to scan all the rules until a match is found. This approach becomes inefficient if the number of rules is too large and if the majority of the packets match with rules located at the end of the rule set. In this thesis, we propose an intelligent predictive technique for packet inspection based on data mining. We consider each rule in a rule set as a 'class'. A classifier is first trained with labeled training data. Each such labeled data point contains packet header information, packet content summary information, and the corresponding class label (i.e. the rule number with which the packet matches). Then the classifier is used to classify new incoming packets. The predicted class, i.e. rule, is checked against the packet to see if this packet really matches the predicted rule. If it does, the corresponding action (i.e. alert) of the rule is taken. Otherwise, if the prediction of the classifier is wrong, we go back to the traditional way of matching rules. The advantage of this intelligent predictive packet matching is that it offers much faster rule matching. We have proved, both analytically and empirically, that even with millions of real network traffic packets and hundreds of rules, the classifier can achieve very high accuracy, thereby making the IDS several times faster in making

matching decisions.

Keywords: Snort, Data Mining, Classification, Security.

Title and Abstract (in Arabic)

تحسين أداء نظام التحقق والفحص الشبكي بواسطة تنبؤ نوع الاتصال الشبكي عبر التصنيف الذكي

الملخص

الهدف هذه االأطروحة هو دراسة الدور الذي تلعبه أنظمة الدفاع الرقمي عبر الشبكات الإلكترونية، طرق التحقق من هوية الاتصال الشبكي وكيفية تحديد الحزم الخبيثة وكشفها وفحص سرعة الاستجابة وتطويرها من خلال تحسين الية البحث المستخدمة حاليا.

ان أهمية أنظمة التحقق وفحص البيانات المرسلة تعد من أهم طرق تتبع الهجمات الالكترونية والتصدي لها ولكن مع تزايد حجم البيانات المرسلة قد لا يتمكن النظام من فحص جميع البيانات المرسلة مما يؤدي الى احتمالية عبور بعض الهجمات الإلكترونية الى وجهتها. تتناول الدراسة تطوير قدرة أداء الفحص لنظام التحقق من الحزم الشبكية بواسطة تصنيف الاتصالات الشبكية والتنبؤ بنوع الاتصال الشبكي.

تحتوي هذه الرسالة على بعض خوارزميات التنبؤ والتي تم استخدامها لتصنيف الاتصالات الشبكية وتحديد الاتصال الشبكي الخبيث مع تحديد ردة الفعل المتوقعة من نظام الفحص.

مفاهيم البحث الرئيسية: امن الشبكات – امن المعلومات – فحص الشبكة – الجدار الناري – اداء الجدار الناري

Acknowledgements

Firstly, I would never have been able to finish my thesis without help from Allah, as when we don't know what we need to improve ourselves, Allah pushes us to interact with good people in order to select the best way to achieve our goals.

I would like to thank my advisor, Dr. Mohammed Mehedy Masud, for his continuous support of my master thesis. I would like to express my gratitude for his patience, knowledge and motivation, which improved my research skills. Dr. Masud taught me a lot of new things; he helped me accomplish and complete several difficult tasks, and supported me when I faced pressure from research results and deadlines.

I would also like to thank Dr. Zouhair Trabelsi, as he helped me register my thesis with Dr. Masud. His advanced network security class helped me to clearly understand and complete the security part of my thesis.

Dedication

To Omer Almaleki, my Mom, my Sister & my beloved wife

Table of Contents

List of Tables

List of Figures

List of Abbreviations

IDS	Intrusion Detection System
HIDS	Host-Based Intrusion Detection System
NIDS	Network-Based Intrusion Detection System
IP	Internet Protocol
DoS	Denial of Service
DDoS	Distributed Denial of Service
SID	Snort ID
RTN	Rule Tree Node
OTN	Option Tree Node

Chapter 1: Introduction

1.1 Overview

Security systems that monitor network packets, such as firewalls and IDSs, should be able to hold, analyze and log a network packet, as well as apply the required rules rapidly. Advanced attacks take advantage of a firewall's or IDS's performance issues; therefore, security researchers focus on both high performance and efficiency.

Data mining is used to enhance the performance of several applications, and predicts the correct decision based on a training data set. Therefore, data mining improves both security and performance by predicting the right rule, instead of checking every possible rule.

1.2 Problem Statement

The large number of attack signatures makes IDS match and compares a lot of rules with incoming and outgoing packets. This mechanism of rule–packet comparison is accurate, but it may not work properly with a large number of packets and a large number of rules, as each rule is checked one by one. For example, if we received a malicious packet and this packet matched the last rule, then the IDS would take a long time to produce this matched rule.

This would negatively affect an IDS's performance, and may result in errors, allowing some packets to pass into the network without analysis and detection. Therefore, an IDS should find the right rule as fast as possible, without any performance issues.

1.3 Motivation and Contribution

IDS rules have increased, and many other devices instead of just a laptop or PC can now be used to send malicious packets or attack others. The integration of physical and network penetration activities extends the scope of attacks, rules and network traffic.

The options of now running attack tools with smartphones or installing an operating system such as Kali (which provides built-in common penetration test tools) are available for everyone. IDSs and other defense systems should provide high performance to secure networks and the Internet of things and smart cities' systems. We use data mining to enhance the IDS's performance by classifying malicious packets with corresponding rules, and sending those rule to IDS for verification against the received rule.

1.4 Relevant Literature

IDSs apply a packet matching technique that is similar to the packet filtering technique used by firewalls. In addition, IDSs apply signature matching to connect signatures with packet content. We will first discuss some relevant work in packet filtering enhancement, before moving on to discuss deep packet inspection by IDSs. Most of the existing research on the performance of firewalls focuses on the improvement of packet searching times by using various mechanisms, including hardware-based solutions (Baboescu, 2001), specialized data structures (Goyal, 2015) (Woo, 2000), and heuristics (Gupta, 2001). Research works in (Hamed, 2006) (Kencl, 2006) focus on statistical filtering schemes to improve the average packet processing time. The structure of searching by taking into account packet flow

dynamics has been introduced by (Kencl, 2006), (Acharya, 2007). The segment-based tree search (STS) scheme (El-Atawy, 2007) uses bounded depth Huffman trees to enhance the search based on statistics collected from segments. The idea of firewall optimization through early packet rejection was introduced by (Mothersole, 2011), (Trabelsi, 2011) – (El-Atawy, 2009). In (Trabelsi, 2011), early packet rejection is done through rule-fields ordering. In (Trabelsi, 2012), early packet rejection is done through a multilevel filtering process that includes field and intersection filtering modules. In (Mothersole, 2011), an approach named FVSC is proposed to optimize the rejection path. This technique uses a set cover approximation algorithm to construct early rejection rules from original security policy common field values. The PBER technique introduced by (El-Atawy, 2009) is considered a generalization of FVSC (Mothersole, 2011), in the sense that FVSC only focuses on a rejection path while PBER finds shortcuts for both accepted and rejected packets. There has been some work on rule-filtering optimization using data mining. For example, (Cohen, 2005) applies a decision tree classifier for packet classification. In this case, the class label is either ‘accept’ or ‘deny’. However, our approach in this study addresses the problem differently: here, each class label is a rule rather than an ‘accept’ or ‘deny’. In our previous work (Mustafa, 2013), we applied a data mining technique to enhance packet filtering. However, this current work is more challenging, as the previous study only dealt with the packet header, which consists of a small number of features. On the other hand, this current, proposed work deals not only with the packet header, but also with packet content. Therefore, the number of features is very large (e.g. consider each byte of content as a feature) and the learning is more complex. However, we have applied a heuristic to reduce the feature set and improve learning performance and accuracy.

Chapter 2: Background – IDS and Data Mining

2.1 Information Security

Information security provides a set of policies and systems that protect information from unauthorized people.

Protection means that unauthorized people are unable to access or modify data, even by accident. Protection mechanisms, systems and algorithms are created to preserve information confidentiality, integrity and availability.

Confidentiality refers to the protection of data from unauthorized access, and the provision of access to authenticated users only. An example of a protection mechanism is access control implantation, which is one of the security mechanisms used to achieve confidentiality.

Examples of confidentiality attacks:

- Session hijacking and the theft of user credentials.

Integrity refers to the protection of data from unauthorized modification. Integrity is different from confidentiality. For example, providing access to students so they may see their grades does not mean that they can change their marks. Cryptography techniques such as hashing algorithms and digital signatures are examples of mechanisms used to achieve the integrity goal.

Examples of Integrity attacks:

- Data modifications through a man-in-the-middle (MITM) attack.
- Data modification through SQL injection and cross-site scripting.

- Data modification using the exploitation phase (e.g. leveraging the application server privileges with its default credentials).

In terms of availability, the required system, application, software, database, hardware and other assets should be available when requested; however, it does not mean that the user can request these assets 24/7. User–service interaction should be regulated by policy rules.

Availability is a very critical issue, as it does not make sense to preserve both confidentiality and integrity for data that is not available to anyone! One single interruption may cost the organization a lot of money. Load balancers, backup systems, the offline mode, and the installation of a UPS battery in the data server room are examples of availability systems.

Examples of availability attacks:

- Physical destruction that makes the servers shut down, or causes the disconnection of the database or any other network component.
- Distributed denial of service (DDOS).

2.2 Detection and Prevention Systems

Network attacks have become the weapons used by criminal organizations and malicious groups. People participate in social networks, transfer their credit card data, pictures and videos, and share their personal locations and other sensitive documents with each other online; this motives a lot of criminals and malicious groups to hack into the Internet.

Security breaches may cause a lot of damage to victims. Intruders are everywhere, threatening and penetrating organizations; network administrators set several defense processes and systems to prevent intrusion attempts. However,

prevention is only a single line of defense in the face of these malicious attempts. Detection is also required to improve an organization's security.

Detection catches what prevention misses. For example, it is possible to detect firewall bypass attempts, access control bypasses (privilege escalation), or simply trace someone's activities. Various network security mechanisms can counter attacks, and these mechanisms include firewalls, IDSs, IPSs and honeypots. Each of these mechanisms provides a different service to counter a threat and mitigate the risks. For example, the role of a firewall is the prevention of malicious traffic, whether that be software, hardware or both. It contains predefined rules for malicious attempts and allows users to create their own security rules, grouping them together into a security policy that it then applies. Firewalls can be personal (for a single user) or can be an enterprise's firewall (for an organization's network). Palo Alto Networks, Juniper Networks and the Cisco ASA firewall are examples of commercial firewalls.

IPS is a combination of both IDS and firewall functionalities. IPSs have most of the detection and log techniques offered by IDSs. Moreover, an IPS can prevent what it detects. An example of an IPS is the Cisco IPS 4200 series.

Honeypots are vulnerable systems. These offer a lot of vulnerability to the attackers in order to trap them and trace their activities, such as security control bypasses, privilege escalations, and so on. The data generated is then used to improve the current security system.

An IDS is a detection technique used to monitor, alert and log suspicious traffic. Host-based IDSs (HIDSs) and network-based IDSs (NIDSs) are different types of IDSs.

A signature-based method and an anomaly-based method are the different methods for detection. Both detection methods allow the system administrator to create his own rules and execute them with specific users, throughout the subnet or throughout the whole organization's network.

2.2.1 HIDS

An HIDS is used to monitor the user's PC for misconfigurations, policy enforcement, rootkit detection, integrity, event correlation and log analysis. It is useful to monitor those audit trails that determine an insider or a policy violation, as well as to trace the improper activities of a specific user ID. Due to its position, it can identify malicious activities over encrypted networks or switched network topology. Some useful HIDS services are:

File integrity checking (berkeley, n.d.): Generate periodically cryptographic checksum value to maintain the integrity of files.

File attributes checking: Check file permission and ownership modifications.

File access attempts: Monitor file access for both users and applications, and the type of requested access (e.g. read, write or execute).

Code analysis: Monitor and check the attempt of execute code, such as buffer overflow attacks. This is useful to thwart privilege escalation, malware and unauthorized access (google, n.d.).

Network configuration monitoring: Monitor the integrity of the network configuration of a host.

The drawbacks of HIDSs are:

Draw on the resources of a user's host.

Cannot detect a packet over the network (sans, 2005).

Tripwire is an example of an HIDS created by Dr. Eugene Spafford and Gene Kim in 1992 at Purdue University.

2.2.2 NIDS

An NIDS is used to monitor an organization's network to prevent malicious traffic. It can use one of the common detection methods (signature-based detection or anomaly-based detection).

Both detection methods have advantages and disadvantages. For example, signature-based detection cannot detect zero-day attacks, while an anomaly-based detection system can. However, an anomaly-based method generates more false positive alarms than a signature-based method.

2.2.3 IDS Detection Approaches

Signature-based detection:

This approach first sniffs incoming or outgoing network traffic, and then compares these sniffed packets with a set of rules in order to identify a malicious packet. Additional configurations include spans or a mirror port in a network switch required to see all types of network traffic. An NIDS network's position is very important – for example, placing an IDS as a first line of defense in order to monitor a firewall's performance is a good idea.

Anomaly-based detection:

Anomaly-based detection requires security administrators to identify unexpected behavior. Examples of unexpected behavior include an ICMP packet with a large payload size, or sending a large number of packets with an SYN flag to well-known ports to specific PCs.

Signature-based detection:

These systems detect malicious traffic based on a unique pattern – for example, a path traversal or directory traversal attack should contains dots and slashes (../../../../); therefore, the signature of the directory traversal attack is (../../../../). Another example is if an attacker sends an http login request with ‘admin’ as the username and the wrong password more than a specific number of times, then these admin login attempts will be logged and the IDS will send an alert. The ‘admin’ keyword along with the wrong password is an example signature (admin login attempt). Signatures are stored in an IDS database and are compared with each network packet.

2.3 Snort

2.3.1 Introduction

Snort is an open-source packet sniffer and IDS created by Martin Roesch in 1998. It was created to sniff network packets such as tcpdump. Later, it was improved to detect malicious packets and identify attacks, which it can now do both over the network and offline (by reading the pcap files).

Snort is a signature-based analysis. Each Snort rule has content that represents a unique pattern present in a malicious packet. Some packets can match with more than one attack signature – for example, the first alert may be generated if an attacker requests the website's admin page, and the second alert could be generated due to a failed admin login attempt.

2.3.2 Configuration

Various rules, such as web-attacks, SQL injection, scan attempts, virus, bad-traffic, ftp attacks and other rules, are written by the Snort community. Snort's configuration file activates and maps these rules with certain important variables, such as the following.

HOME_NET: This is a network you need to protect from outside attacks. It accepts IP network addresses or can also accept 'any' as a string to detect all malicious packets.

External_NET: This is a network that is outside the scope of the home network (e.g. the Internet, a third-party network). It also accepts 'any' as an input to detect everything. The (#) hash symbol to comment the rule. The configuration file allows users to modify and enter all the details of their internal and external servers, as well as configure major detection components, such as a decoder, a preprocessor and an output plugin.

2.3.3 Snort Architecture

Packet decoder:

A series of decoders is used to decode or reconstruct a network packet to prepare it for the rest of the IDS's components. Each decoder responds to a specific network layer. For example, when a decoder receives a packet, it checks what physical interface the packet contains and sends the packet to a specific decoder that responds to that interface, after which it checks what transport protocol packet is used and then sends it to the right decoder. Various decoders are used for different interfaces (Caswell, 2007).

Layer protocols and decoders:

Interfaces such as the Ethernet have their own decoders. An 802.11 has its own decoder, and network layer protocols (such as IP, ARP and IPX) have their own decoders. An ICMP decoder, TCP decoder and UDP decoder are responsible for decoding a packet's transport layer. Figure 1 illustrates the relationship between a decoder and the network layers.

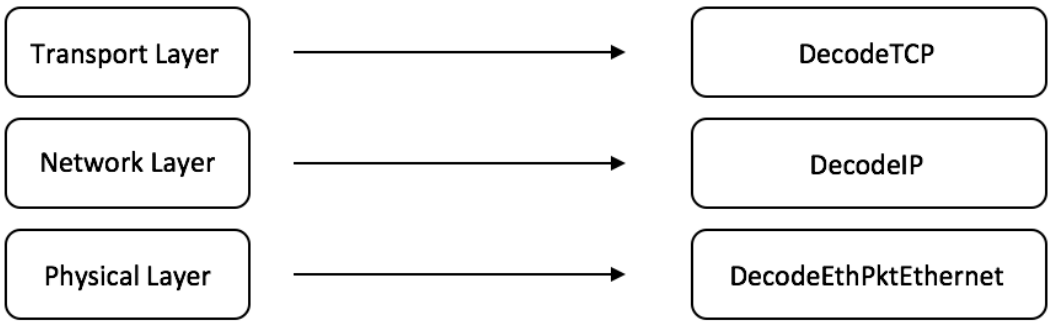


Figure 1: Layers and decoders

Decoders have their own rules during the packet decoding process. A decoder rule is generated when an error occurs; for example, if there is invalid IP header length with a received packet, then the decoder will log that error and create an alert.

```
alert ( msg:"DECODE_IPV4_INVALID_HEADER_LEN"; sid:2; gid:116; rev:1; metadata:rule-type decode;
classtype:protocol-command-decode; )
```

Figure 2: Snort rule

The decoder decodes a packet by reconstructing its original structure into a specific structure. It set pointers for the most crucial aspects in terms of detection, and allows components such as the preprocessor and the detection engine to gain a clear picture of their targets inside the packet (Caswell, 2007). In short, it prepares the scene of investigation for the preprocessor and detection engine.

Preprocessor:

This normalizes the network packets for the detection engine. For example, if the preprocessor receives a fragmented packet, it will wait until it receives the full packet before sending it on to the detection engine. Preprocessors are very important for successfully completing the detection process, as they prepare a packet payload for the detection engine. Certain special preprocessors are used to analyze an incoming packet to help identify those attacks that have no signature. For instance, there is no content that describes a port sweep attempt; however, new preprocessors use a technique called target-based detection that reassembles the packet and segments as a target. Preprocessors also prevent issues in terms of fragmented packets. If an attacker has sent a malicious packet to a Windows operating system, Windows will arrange those packets in a specific order that is different from a Linux

operating system. This means that an IDS should order the fragmented packets according to a Windows operating system and not a Linux operating system.

Rules:

In a detection process, each rule is split into two sections: the first section is called a rule tree node (RTN) or rule header, and the second is called an option tree node (OTN) or rule option.

RTN data contains a required action, which is an action that is generated if a packet matches that rule. Examples of action values are activate, dynamic, alert, pass and log, and examples of protocol values are TCP, UDP, IP or ICMP. The home network IP address, port number, and the direction symbol are used to first identify the source and destination (<>), and then the external network and its port number. OTN data contains a message that will be displayed in the Snort log (MSG), a Snort rule identifier (SID), an attack signature (Content), rule classification (Class type) and so on (snort, n.d.).

Detection Engine:

This compares network packets with each possible rule. The comparison process has initial requirements. For example, Snort should divide each rule to two tree nodes (rule header and rule option), after which it will split the incoming packet into another two sections (packet header and packet payload). The packet header will be compared with the rule header until a match found, and then the packet payload will be compared with all possible rule options for the selected rule header. Snort will generate the required action mentioned in the rule header only once a packet

header has been matched with a rule header and a packet payload matched with a rule option.

The detection engine is the core of the comparison process. It receives packets from the preprocessor and checks if there is any malicious content in a packet payload by comparing it with attack signatures (content). The detection engine receives packets, checks the protocols contained – these can be TCP, UDP, IP or ICMP – and then selects the root node based on those four protocols. So if the received packet contains an IP protocol and an TCP protocol, then Snort will check both protocols and the packet will check both RTNs.

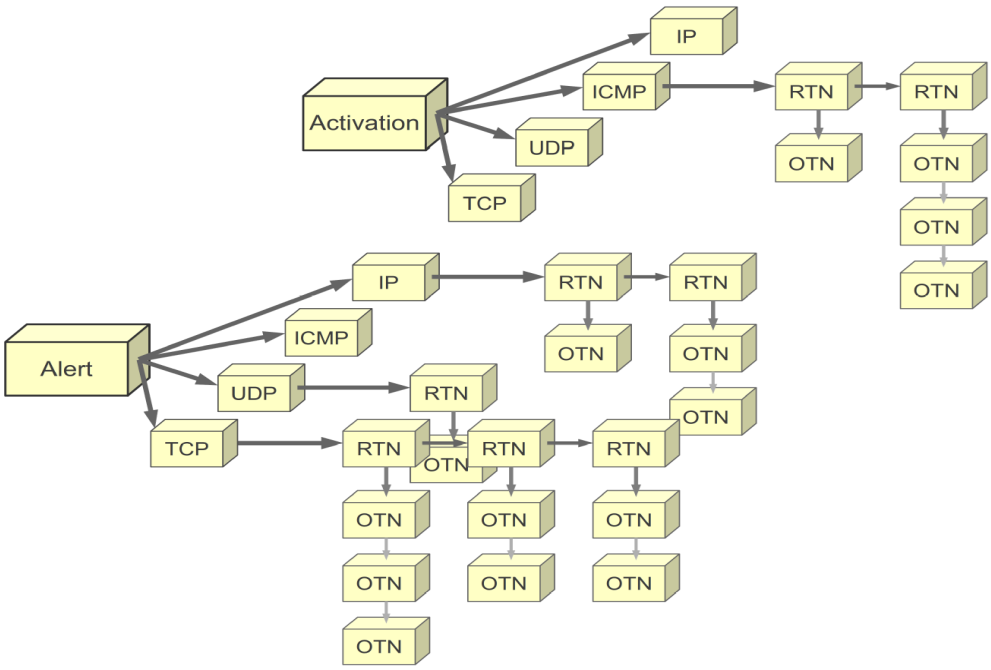


Figure 3: Snort constructs detection rules

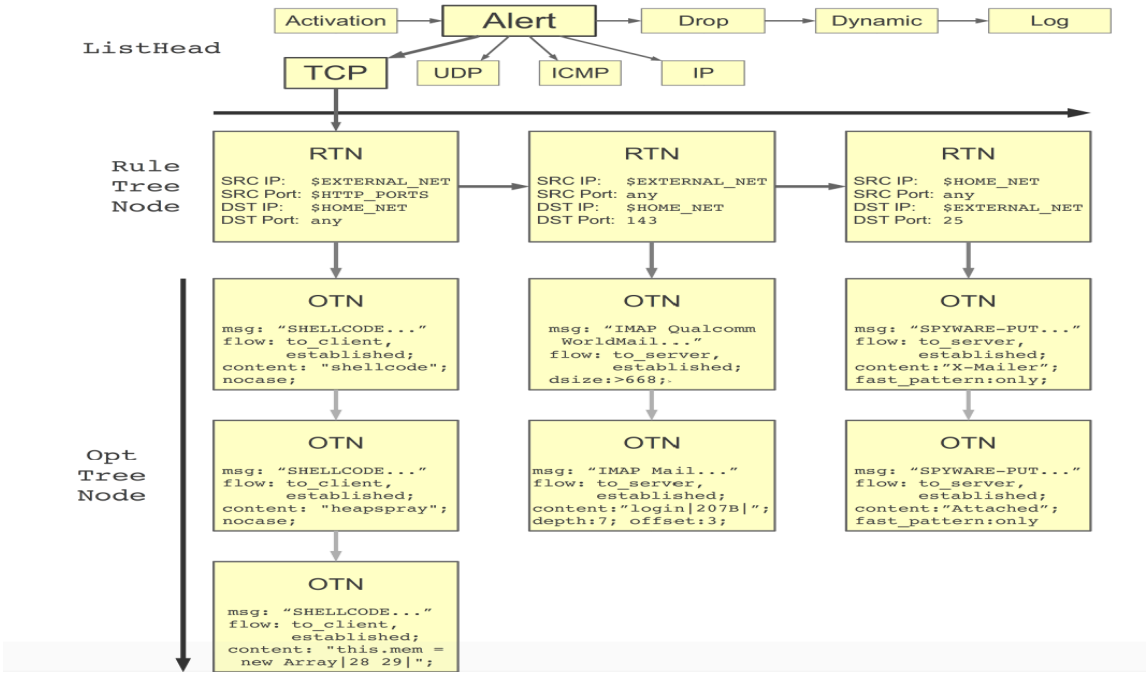


Figure 4: Snort RTNs and OTNs

Output Models:

Several output plugins and formats are used to present Snort results. CSV, XML and PCAP are examples of output formats.

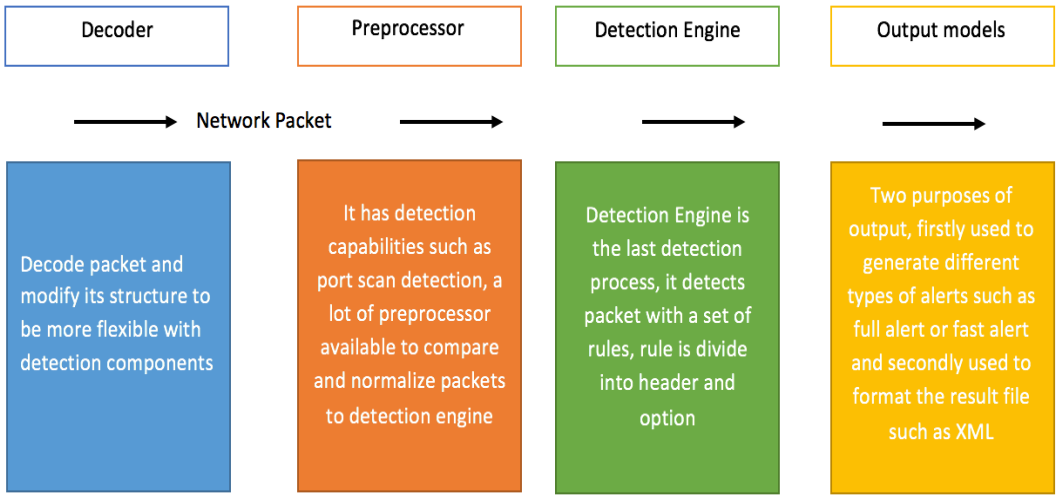


Figure 5: Snort architecture

Snort Content:

Snort is a signature-based IDS that relies on a large set of signatures. SID is a unique snort ID for IDS signatures, and each rule can have fragmented pieces of single signature. Not all rules have content. Some rules can generate an alert if there is a specific IP address in the packet header. Alternately, if the packet header has encryption data and a specific port (such as an SSH) is used, there is nothing in the payload to analyze and no content in the payload will match the rule options.

However, the rule option may have one or more pieces of content, such as the following example.

alert tcp $EXTERNAL_NET any -> $HOME_NET 139 (msg:"NETBIOS SMB wkssvc unicode little endian andx bind attempt", flow:established,to_server, **content:"|00|",** depth:1, **content:"|FF|SMB",** within:4, distance:3, pcre:"/^(\x75|\x2d|\x2f|\x73|\xa2|\x2e|\x24|\x74)/sR", byte_test:1,&,128,6,relative, content:"%", depth:1, offset:39, byte_jump:2,0,little,relative, **content:"&|00|",** within:2, distance:29, byte_jump:2,-6,relative,from_beginning,little, pcre:"/^.{4}/sR", **content:"|05|",** within:1, byte_test:1,&,16,3,relative, content:"|0B|", within:1, distance:1, **content:"|98 D0 FF|k|12 A1 10|6|98|3F|C3 F8|~4Z"**, distance:29, flowbits:set,dce.bind.wkssvc, flowbits:noalert, metadata:policy balanced-ips drop, policy connectivity-ips drop, policy security-ips drop, service netbios-ssn, classtype:protocol-command-decode, sid:8900, rev:5.

Moreover, content values may be in the ASCII format, HEX format or both ASCII and HEX formats.

Table 1: ASCII content

CONTENT	**Rule SID**
MIT-MAGIC-COOKIE-1	1225
/sensepost.exe	989
/fp4areg.dll	1247
GET /../../../../../../../../../../	1049
fp30reg.dll	1246

Table 2: : Hex content

CONTENT	Rule SID
B4 B4	163
28 00 01 00 04 00 00 00 00 00 00 00	2124
C2 C5 CD C4 FD F9 FF 86 E4 9A F8 FF E5 9B 98 E5 FC E1 FD A9 FC	6024

Table 3: Content with both ASCII and HEX formats

CONTENT	Rule SID
Insane Network vs 4.0 by Suid Flow|0A 0D|www.blackcode.com|0A 0D|[r00t]|23|	3015
Content-Type|3A| application/x-icq	1832
Proxy-Authorization|3A| NTLM	12362

A Snort rule can identify the position of content in the payload using depth keywords. For example:

```
alert ip $EXTERNAL_NET any -> $HOME_NET any (msg:"EXPLOIT Cisco NHRP incorrect
packet size", ip_proto:47, content:" |01|", depth:2, offset:2, content:"|FF FF|", depth:2,
offset:14, reference:bugtraq,25238, reference:cve,2007-4286, classtype:attempted-user,
sid:12299, rev:2,)
```

Figure 6: Example of depth keyword

Snort Pattern Matcher:

The Snort pattern matcher groups OTNs with a single RTN. A single RTN usually has many OTNs, and a single packet header can send more than one malicious payload. The pattern matcher is used to reduce the number of rules that must be handled, as the number of rules always increases when processing a large number of packets. Snort users can develop their own rule and save it in a local.rules file with a higher SID number to avoid any collision with other SIDs (Snort cannot run with two rules that contain the same SID, as the rule parser gets the rule file from snort.conf and checks the validation of all rules at Snort initialization).

The Parser.c file has many functions that are used for Snort initialization. For example, function ParseRulesFile() is used to prepare all rules from the config file to function ParseRule (). ParseRule () verifies the rules and checks if there are additional instructions that relate to the preprocessor and output plugins; if there are, it provides the required function for each of them (Andrés Felipe Arboleda, 2005). Rule categories are based on protocol type, for example a ListHead structure is used to organize the rules with their action (ferryas.lecturer.pens.ac.id, n.d.).

```
typedef struct _ListHead
{
    RuleTreeNode *IpList;
    RuleTreeNode *TcpList;
    RuleTreeNode *UdpList;s
    RuleTreeNode *IcmpList;
    struct _OutputFuncNode *LogList;
    struct _OutputFuncNode *AlertList;
    struct _RuleListNode *ruleListNode;
} ListHead;
```

```
ListHead Alert;         /* Alert Block Header */
ListHead Log;           /* Log Block Header */
ListHead Pass;          /* Pass Block Header */
ListHead Activation;    /* Activation Block Header */
ListHead Dynamic;       /* Dynamic Block Header */
ListHead Drop;
ListHead SDrop;
ListHead Reject;
```

Figure 7: Both sides show the list of protocol types, output types and action types

Two different structures are used for both RTNs and OTNs, and ProcessHeadNode() is used to call and prepare OTNs for each RTN. An RTN structures header details, while an OTN stores rule options.

Fast Pattern Matching Algorithm:

Snort uses many string matching algorithms to ensure that the packet contains content (such as the Boyer–Moore (Fisk, 2002) and the Aho–Corasick (Fisk, 2002) algorithms). The Aho–Corasick algorithm is based on a finite state machine: the algorithm should have a set of keywords that can be compared with a given text, and it can search multiple patterns simultaneously. The Boyer–Moore algorithm, however, uses two methods to match or find the pattern in a text. The first method is to construct a bad match table to match a given pattern with the text.

2.4 Data Mining

Data mining and predictive analysis has improved the performance of many systems. Applications use data mining techniques to improve their performance, as

security systems (whether a prevention system, such as a firewall, or a detection system, such as an IDS) require high performance to monitor network packets. Both prevention and detection systems should not allow network packets to bypass the required detection due to performance issues. Banking systems use data mining for consumer credit cards to improve their offers (i.e. they predict consumer purchase patterns based on specific purchases) and to predict future financial risk. WebWatcher is a data mining application used to create adaptive websites that automatically improve their presentation based on the user's access pattern.

Data mining comprises different techniques. However, due to the study's scope, this thesis scope will focus on the predictive model of a decision tree and on classification techniques.

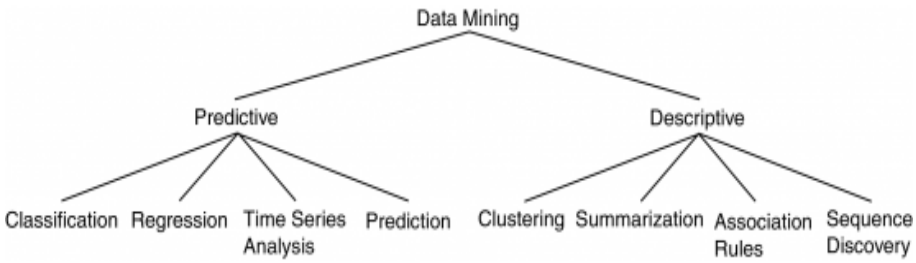


Figure 8: Data mining tree

Classifiers are used to classify the input and give it an appropriate class. Training data should be made available to classifiers for them to learn from it. For example, the training data may include malicious packets and their corresponding rules, which allows the classifier to learn how to act with such future inputs.

2.4.1 Naïve Bayes Classifier

A naïve Bayes classifier is based on Bayes' theorem, which is easy to understand. Bayes' theorem uses prior probabilities and the likelihood of classification based on adjacent patterns. There are two prior probabilities in Bayes' theorem: the first prior probability is for class, while the second is for object (dell, n.d.).

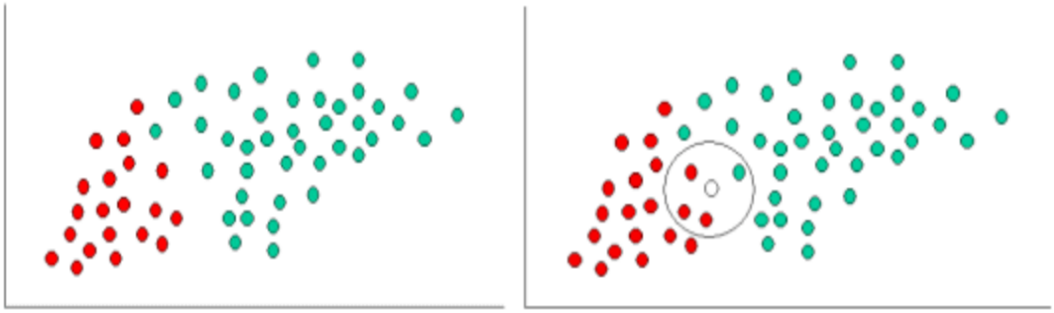


Figure 9: A naïve Bayes classifier. The left side describes class prior probability and right side is the likelihood of a new object based on its position

Figure 9 shows two classes, green and red. The green class is more than double the red class; thus, we assume that the probability of a new object belonging to the green class is greater than the probability of it belonging to the red class. This assumption is called prior probability. Prior probability is an assumption based on prior observation and is a part of Bayes' theorem. The second part of Bayes' theorem is the likelihood of an object belonging to a class. For example, in Figure 9, the red objects appear in the left side of the shape's space, while the green objects occupy the right side. Therefore, we can assume that a new object on the left side is more likely to classify as red than to classify as green, and vice versa.

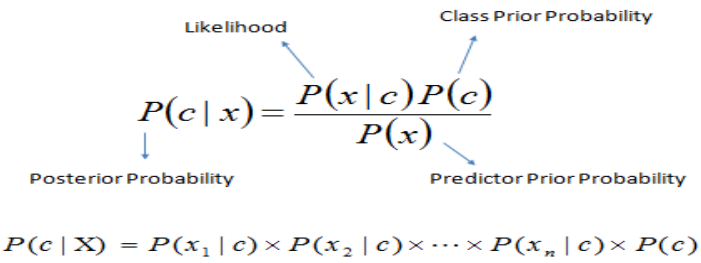


Figure 10: Posterior probability

2.4.2 Decision Tree

A decision tree is a learning method used in machine learning, and decides the output by constructing a tree of given inputs. Various algorithms are available to build decision trees, such as Iterative Dichotomiser 3 (ID3) and C4.5. A decision tree builds tree nodes of the given inputs, and arranges instances from root nodes to leaf nodes.

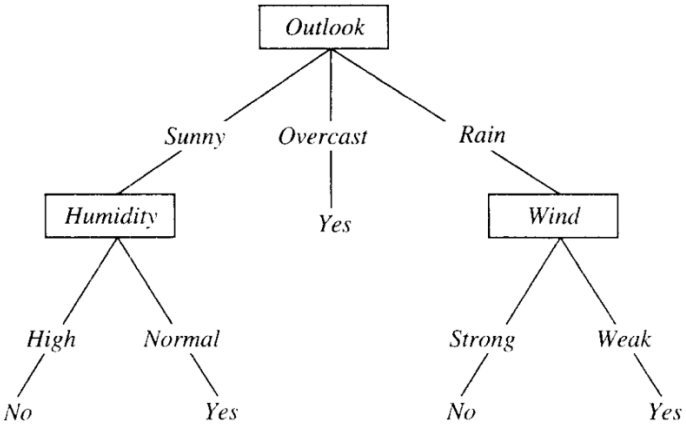


Figure 11: Decision tree

In a decision tree, the 'leaves' are always decisions. Figure 11 discusses the possibility of playing golf with different weather situations. There are many situations with two decisions (classes) yes or no. For example, the player can play

golf if the outlook is sunny and the humidity is normal, or if we use if-then rules with an overcast outlook, the result will be:

If ‘outlook = overcast’ then ‘play’.

ID3:

ID3 is a learning method that uses both entropy and information gain within a data set to split the original set and generate a node tree. ID3 selects the root of the given data set based on information gain score, with the highest score acting as the root of the decision tree. The method keeps splitting the tree nodes until a leaf node is found.

C4.5:

C4.5 has more features than ID3, and both classification algorithms construct the tree nodes differently. ID3’s limitation is its overfitting problem; C4.5 solves this overfitting problem through the pruning technique.

2.4.3 N-gram analysis

N-gram is an important part of language modeling. N-gram uses the probability of the prior values to predict the new value; for example, it can predict the availability of a word in a sentence based on the prior probability of that sentence.

Unigram:

A unigram is an estimated likelihood of a word (or character) occurring in a given text (or word) based on the frequency of occurrence.

UNIGRAM = U, N, I, G, R, A, M

Bigram:

A bigram is an estimated likelihood of two contiguous words occurring in a given text based on the frequency of occurrence.

BIGRAM ≈ BI, IG, GR, RA, AM

Trigram:

A trigram is the estimated likelihood of three contiguous words occurring in a given text based on the frequency of occurrence.

TRIGRAM ≈ TRI, RIG, IGR, GRA, RAM

Chapter 3: Proposed Technique

The proposed Snort (based on data mining) will be referred to as the 'Intelligent Predictive Packet-Inspect Snort' (IP^2S). The high-level overview of the IP^2S is illustrated in Figure 12. There are two main components of the Snort, namely the offline and the online components. The offline component collects network packets and uses them to train a classifier. The online component does the actual packet filtering online. Here, the classifier trained with the offline component is used to classify each incoming packet. The predicted class corresponds to a rule, which belongs to a set of filtering rules. If the classifier predicts the correct rule, i.e. the predicted rule matches the packet, then the corresponding rule action is taken. Otherwise, the prediction is wrong and, in that case, the traditional Snort is used to find the matching rule for the packet and the corresponding rule action is taken. By 'rule action', we mean the action (e.g. alert) corresponding to the rule.

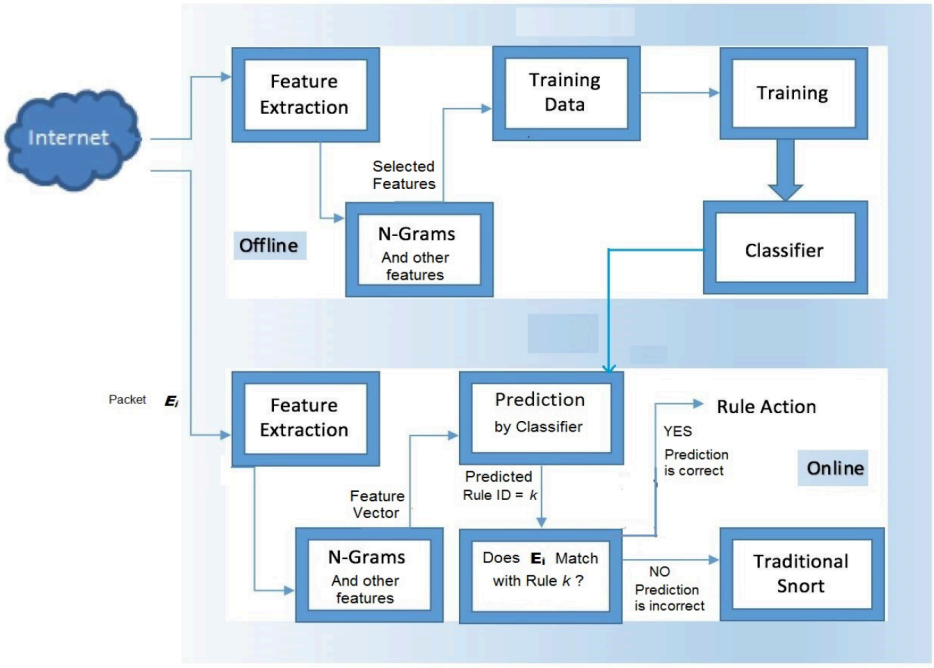


Figure 12: Proposed technique for the IP^2S

Before going into the details, we will introduce several terms that will be frequently used in this chapter.

Definition 1 – Packet (Ej): A network packet Ej is a data structure consisting of two parts: the header and the payload, denoted by H(Ej) and L(Ej) respectively.

The header contains attributes such as protocol, source IP, destination IP, source port, destination port, and so on. The payload contains binary data (i.e. the content of the packet).

Definition 2 – Feature set (F): The feature set is a set of attributes or features that is used for training and classification. The feature set consists of two subsets: the header feature set (H(F)) and the payload feature set (L(F)). The former is extracted from packet headers and the latter is extracted from packet payloads.

Definition 3 – Feature vector (V(Ej)): The feature vector V(Ej) is a vector of feature values for the packet Ej, corresponding to the feature set F.

Definition 4 – Rule (Ri): A rule Ri consists of three parts. The first two parts consist of the rule header and rule options, denoted by H(Ri) and O(Ri) respectively. The rule option section may contain zero or more options. The third part of rule Ri is the rule action, denoted by A(Ri). If a packet header matches the H(Ri) and the content (i.e. payload) matches the O(Ri), then the action A(Ri) is taken. Therefore, formally, a rule can be represented as H(Ri) V O(Ri)) A(Ri). The rule header of a rule consists of a number of field tests (e.g. Source IP = 10.*.*. * AND Protocol = TCP AND Destination Port = 80 AND ...). The rule option usually contains a signature (string or pattern) that must be present in packets, and the rule action in Snort is usually an alert.

Definition 5 – Rule match (M(Ej, Ri)): A packet Ej matches a rule Ri if the field values of the packet header H(Ej) satisfy all the field tests in the rule header H(Ri), and if the packet payload L(Ej) matches all the rule options O(Ri). We will denote this case (i.e. when Ej matches with Ri) with the symbol M(Ej, Ri). Likewise, ¬M(Ej, Ri) will be used to denote cases where Ej does not match with Ri.

Definition 6 – Rule set (R): The rule set R = {R1, R2, ….., RN} is the set of N rules in the Snort, where each Ri is a rule in the set.

Definition 7 – Standard Snort (SSn): The SSn is an IDS that, for each incoming packet Ej, sequentially searches through the rule set R, starting from the first rule R1. If Ej does not match Ri (i.e. ¬M (Ej, Ri)), then the search continues with the next rule Ri+1. Otherwise, if M (Ej, Ri), the action A(Ri) is taken.

3.1 Feature Extraction and Selection

We extract two types of features from packets: features from the packet header and features from the packet content (i.e. payload).

1) Packet payload features: Packet payloads contain binary data. We use the N-gram feature extraction technique to extract features from the payload. An N-gram is a sequence of N consecutive bytes in the payload. For example, if you assume that 0304051B1D1EF2F3F4FEFF is the payload (hex values), then the one-gram (one-byte) sequences will be 03, 04, 05, and so on. The two-gram (two-byte) sequences will be 0304, 0405, 051B, and so on. Likewise, the three-gram (3-byte) sequences will be 030405, 04051B, 051B1D and so on. Note that there are 256 (= 28) unique one-grams possible, and there are 65,536 (= 216) possible values of two-grams and so on. Therefore, there are 28N possible different N-grams, which is a very large number for large values of N.

a) Generating a one-gram feature vector: For one-gram, the total number of features is 256 (00, 01, 02, FF), which is a manageable number. We can generate a feature vector for one-gram as follows. For each packet, the feature vector consists of 256 binary values (i.e. 0 or 1). The i-th value in the vector has value = 1 if the packet contains the corresponding feature. For example, given the payload F2 04 F4 05 FE 1B 05 1E 03 FF F3, the feature vector would look like 0 0 1 1 1 0 0 .... 1 (256 values). The first value is 0 because the corresponding one-gram (i.e. 00) is not present in the payload. The second value is also 0 for the same reason. Then we have 1 1 1 because the next three one-gram features (03, 04 and 05 respectively) are present in the payload. The last value (i.e. the 256th value) is 1 because the corresponding one-gram, FF, is also present in the payload.

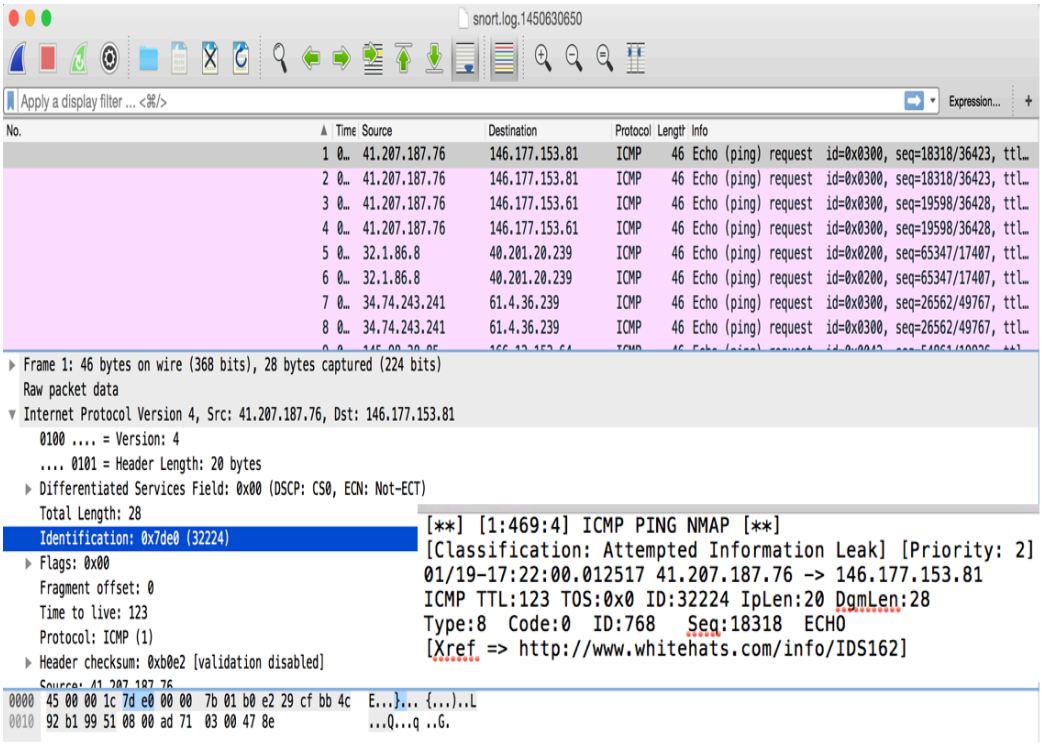


Figure 13: Packet identification that exists in both payload and the corresponding Snort alert

```
36195,1,1,1,1,1,1,1,1,0,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,1,0,0,0,1,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,1,0,0,0,0,1,1,0,1,1,0,0,0,0,0,0,1,0,0,0,0,0,0,
0,0,0,0,1,0,0,0,0,1,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1,1,1,0,0,0,0,0,1,0,0,0,0,0,0,0,
1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,0,0,0,1,0,0,0,1,0,0,1421
```

Figure 14: One-gram feature vectors. The first value refers to packet ID and the last value is a class

b) Generating N-gram feature vectors (N >= 2): As mentioned before, the total number of two-grams is 65,536 and it grows exponentially with the value of N. Feature vectors with such a large number of features will not only take up large amounts of memory but will also have high processing time during training. However, the feature vectors will be very sparse. Therefore, we have devised a

heuristic approach for generating the two-grams or higher features, as explained below.

Note that packet payload is inspected by Snort only if the corresponding rule contains an option with content, i.e. signature. Therefore, instead of considering all possible N-grams as features, we collect the signatures from all rules and generate N-grams from those signatures only. This drastically reduces the total number of N-grams generated. For example, if there are 100 rules and each rule contains a 10-byte signature on average, the maximum number of N-grams would be (at most) 1,000, whatever the value of N. However, we go one step ahead by further reducing the number of N-grams generated by selecting the best K based on information gain.

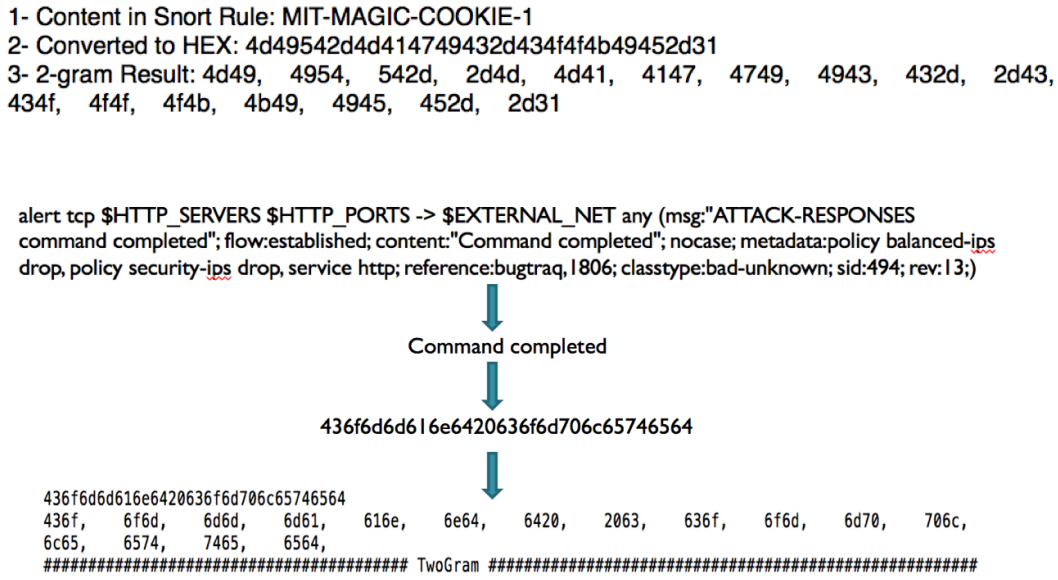


Figure 15: An example of two-gram feature vectors

3) Putting it together: In summary, the feature extraction and selection process works as follows.

Construct a feature set for the packet headers (denoted as (H(F))), which

includes features such as IPSRC, IPDEST, PORTSRC, etc.

Construct a feature set for N-gram features, N >= 1. Apply feature selection to reduce the number of these features. The selected set of features would be denoted as L(F).

The final feature set would look like this: IPSRC, IPDSET, PORTSRC, ..., 1-gram1, 1 gram2, ... ,2-gram1, 2-gram2,

The feature vector V(Ej) (for the above feature set) corresponding to a packet Ej would look like this: 10.100.10.11, 10.11.100.101, 25, ..., 1, 0, ..., 0, 1, ... denoting that the packet's source IP is 10.100.10.11, destination IP is 10.11.100.101, source port is 25, ..., 1-gram1 is present, 1-gram2 is absent and so on.

3.2 Training the IP^{2}S

The training data D = {d_1……., d_M} consists of a set of M training instances d_k, k∈ {1,……..,M}, where d_k is the tuple (V(Ej), i), such that M(Ej, Ri). In other words, each training instance consists of the feature vector V(Ej) for the packet Ej and the class label i of the packet. The class label i of a packet is the index of the first rule (Ri) in the rule set that matches the packet. For example, let us assume the packet Ej matches the 10th rule (i.e. R10) in the rule set. In this case, the class label of Ej is 10. The class label of a packet can be found by running the SSn for the packet. Once we have training data, we can use this data to train a classifier C of our choice. This study has tried many different classifiers, but the best ones in terms of classification time and accuracy of prediction are the 'decision tree' and the 'ripper' classifiers. Note that the whole process of collecting the training data and training the classifier is done offline.

3.3 Filtering with the IP^{2}S

A packet Ej is provided to the classifier C as input. Note that we only provide the packet, not the class label. The class label of a packet is the index of the rule that matches the packet. The task of the classifier is to predict the class label of Ej. Let P(Ej) be the prediction (output or predicted class label) of the classifier for the packet Ej. Let P(Ej) = i, i.e. the classifier predicts i as the class label. Since the classifier prediction can be wrong, we must check the validity of this prediction. Therefore, we now have to test if M(Ej, Ri). If yes, then the prediction P(Ej) is correct and the corresponding action A(Ri) is taken. Otherwise, if the prediction P(Ej) is wrong (i.e. if ¬ M(Ej, Ri)), then the packet must go through the SSn to fetch the matching rule and, accordingly, the corresponding action.

3.4 Performance Improvement

In this chapter, we analytically proved how the performance of the IP^{2}S is much improved from the SSn using the predictive filtering technique. The improvement mainly depends on the quality of the prediction, which can be improved by providing enough training data for the classifier to learn.

Let T_S(Ej) be the time needed to filter the packet Ej with the SSn, and T_D(Ej) be the time needed to filter Ej with the IP^{2}S. In addition, let I_L(Ej) be the indicator function such that:

$$I_L(E_j) = \begin{cases} 0, & \text{if prediction } P(E_j) \text{ is correct} \\ 1, & \text{otherwise} \end{cases} \qquad (1)$$

Let $T_C(Ej)$ be the time needed by the classifier to predict the class label of Ej (i.e. the classification time). Therefore, we can write:

$$T_D(E_j) = T_C(E_j) + I_L(E_j).T_S(E_j) \qquad (2)$$

In other words, equation 2 states that the time taken to filter a packet by the IP^{2}S is equal to the classification time if the prediction is correct, and the classification time plus SSn filtering time if the prediction is wrong. Therefore, the time needed to filter a batch of B packets $\xi = \{E_1 \ldots, E_B\}$ is given by:

$$T_S(\mathcal{E}) = \sum_{j=1}^{B} T_S(E_j) \qquad (3)$$

$$T_D(\mathcal{E}) = \sum_{j=1}^{B} T_C(E_j) + (1-p) \sum_{j=1}^{B} T_S(E_j) \qquad (4)$$

Here, p is the percentage of packets correctly classified (i.e. predicted correct) by the classifier. In other words, p would be the accuracy of prediction for the set of packets E.

Using equations 3 and 4, we can infer that there would be a gain in filtering time if:

$$
\begin{aligned}
& T_D(\mathcal{E}) < T_S(\mathcal{E}) \\
\Rightarrow & \sum_{j=1}^{B} T_C(E_j) + (1-p)\sum_{j=1}^{B} T_S(E_j) < \sum_{j=1}^{B} T_S(E_j) \qquad (5) \\
\Rightarrow & \sum_{j=1}^{B} T_C(E_j) < p\sum_{j=1}^{B} T_S(E_j) \\
\Rightarrow & T_C(\mathcal{E}) < pT_S(\mathcal{E}) \Rightarrow p > \frac{T_C(\mathcal{E})}{T_S(\mathcal{E})}
\end{aligned}
$$

Therefore, there would be a gain in running time if the prediction accuracy of the classifier is greater than the ratio of the total classification time (T_C) to the total SSn matching time. For many classification techniques, T_C would be much less than T_S. For example, for the decision tree classifier, the classification time is less than half the filtering time of SSn (which is empirically justified by our experiments). Therefore, in this case, there would be a gain even if the prediction accuracy is 50% (i.e. half of the packets are incorrectly classified). In real-world scenarios, the accuracy of a classifier is much higher than 50%, provided that it is trained with enough training data. We also derive an interesting relationship between classification accuracy and the running time of the IP^{2}S from equation 5. It can be seen that the filtering time of the IP^{2}S decreases with the classifier's increasing accuracy. This has been confirmed with the empirical evaluations undertaken with the IP^{2}S (in chapter 5).

Chapter 4: Experiments and Results

In this chapter, we describe the data sets and experimental environment, and discuss and analyze the results.

4.1 Data Sets and Experimental Setup

We have used real network traffic from the CAIDA anonymized Internet traces 2013 data set (caida dataset, n.d.) Furthermore, we generated synthetic 'attack' data using the Metasploit framework.

Figure 16: Process of collecting malicious payloads using the Metasploit framework

Competing approaches: In the IP^{2}S, the classifier used is a decision tree. The SSn is used as a baseline.

Parameter settings: We filter exactly the same set of packets for both the IP^{2}S and SSn; this set consists of 10 million packets. For training the classifier, we use 10,000 packets but these training packets are not used in the test set (i.e. for filtering). Moreover, exactly the same set of Snort rules (consisting of 500 rules) is used to test both IDSs. These rules have been generated by hand, and follow standard

security policies observed in our institution.

Hardware and software: The experiments were done on a standalone workstation that had an Intel Core i5 2.4GHz processor with 8GB RAM and a 750GB hard drive. The operating system was Windows 7. For the SSn, we used Snort version 2.9.8.0, and a major part of the IP^{2}S was developed in Java (NetBeans IDE). We have heavily relied on the Weka machine learning API (waikato University, n.d.) for feature extraction and selection and classification.

4.2 Results and Discussion

We evaluate the system based on the total processing time. Figure 17 shows the processing time comparison between the SSn and the IP^{2}S. The x-axis of this graph corresponds to the number of packets processed (in millions) and the y-axis corresponds to the total processing time in milliseconds. For example, at x=10 million, the y values of SSn and SSF are 822001 and 113181 respectively, meaning that the SSn takes 822,001 milliseconds to filter 10 million packets, whereas IP^{2}S takes only 113,181 milliseconds. Thus, IP^{2}S takes about one seventh of the time taken by the SSn. In other words, the throughput of IP^{2}S is more than seven times that of the SSn.

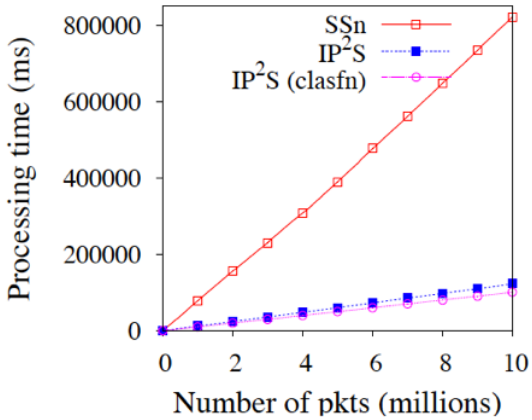


Figure 17: Number of packets vs. cumulative processing time

Figure 17 also shows the classification time (cumulative) taken by the classifier of the IP^{2}S; this is shown with the IP^{2}S (clasfn) curve. It is evident from the chart that the the total filtering time required by the IP^{2}S is only a little more than the classification time, which means that most of the class predictions have been correct. In fact, in this case, the class prediction accuracy was 90% or more. However, based on equation 5 in chapter 4, we can infer that even if the prediction accuracy were as low as 50%, the total running time of the IP^{2}S would be less than that of the SSn. This is because, here, the IP^{2}S's classification time is about one third of the filtering time of the SSn (822 seconds for the filtering time of the SSn and 113 seconds for the classification by the IP^{2}S).

Figure 18 shows how the processing time varies with the number of rules in the IDS. We have run both the IDSs with 50, 100, 200 and 500 rules. As expected, with the increasing number of filtering rules, processing times also increase. But the rate of this increment is higher for the SSn than it is for the IP^{2}S. This is because, with the increase in the number of rules, the time to find a matching rule also increases in the SSn (it has to browse through a longer list). However, for the IP^{2}S, the searching time does not increase as much due to the (mostly) correct predictions

made by the classifier. The slight increase that the IP^{2}S observes is due to the extra time needed to classify an instance (as the tree is now more complex) and the extra time needed when the classifier makes a wrong prediction.

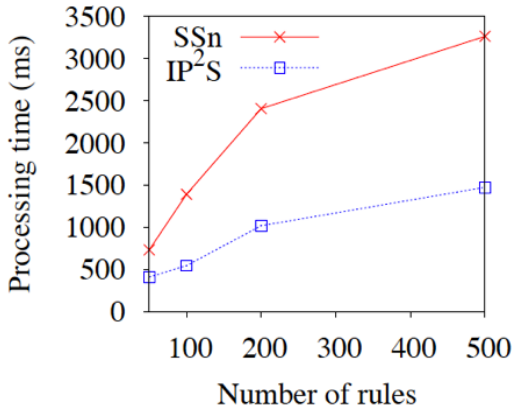


Figure 18: Number of rules vs cumulative processing time

Effect of the number of training data on the processing time and accuracy of the IP^{2}S: The size of the training data has a direct impact on the accuracy of the classifier. Generally, the prediction accuracy of a good classifier should increase with an increasing size of training data. This is observed in our experiments, as shown in Figure 19(a). Here, the x-axis represents the number of training data (in hundreds) and the y-axis represents the prediction accuracy of the classifier on the test data. There are two curves: one representing 100 rules and the other 500 rules. The curve for 100 rules shows the accuracy of the classifier when we have 100 rules in the IDS, and the classifier is trained and tested accordingly. The same description goes for the curve representing 500 rules. Both of these curves observe the expected behavior, i.e. an increase in prediction accuracy with an increase in training data. Recall that, from our theoretical analysis on the impact of classifier accuracy on filtering time (chapter 4), we concluded that the filtering time of the IP^{2}S decreases with increasing accuracy.

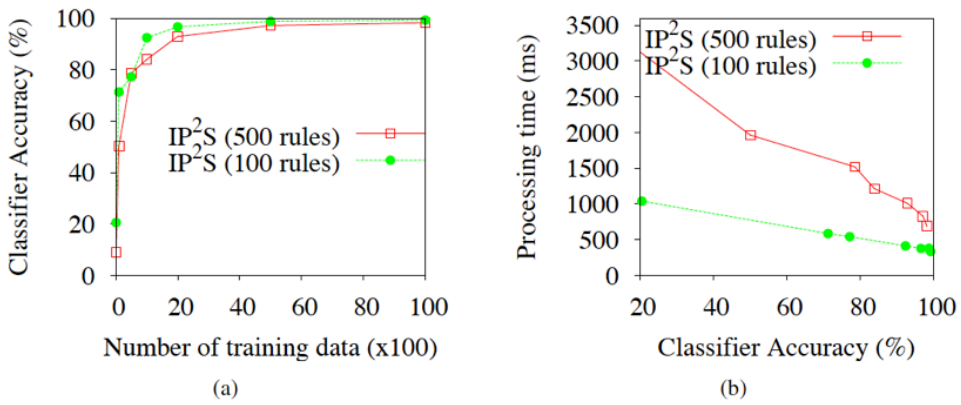


Figure 19(a) size of training data vs classifier accuracy and (b) accuracy vs processing time for the IP^{2}S

The empirical observations confirm this theory, as reported in Figure 19(b). Figure 19(b) shows how the processing time varies with the prediction accuracy of the classifier. In both these figures, we report the performance of the IP^{2}S for 100 rules and for 500 rules. In general, the processing time reduces as the prediction accuracy increases. This is because with higher prediction accuracy, the correct rule is predicted more often, requiring less browsing through the rule list to find the matching rule. We get another interesting observation from these figures.

We observe that, for a larger number of rules, the processing time of the IP^{2}S is higher. For example, when accuracy is 92%, the processing times of the IP^{2}S with 100 rules and 500 rules are 414 milliseconds and 101 milliseconds respectively. Therefore, the larger the number of rules, the higher the running time even for the same rate of accuracy. This happens because, for example, when the accuracy is 92%, only 8% of packets are incorrectly classified, meaning that the IP^{2}S has to browse through the list of rules to find a match for these 8% of packets. However, this browsing takes longer and incurs more hits when the list is larger (i.e. 100 vs 500 rules).

We tested four different classifiers with our two-gram feature data, and found that each classifier has different results. In addition, we tested one-gram feature data with these different classifiers, and the results of the classifiers were less accurate than for two-gram feature data.

J48

Correctly Classified Instances	8337	94.9977 %
Incorrectly Classified Instances	439	5.0023 %
Kappa statistic	0.9199	
Mean absolute error	0.0027	
Root mean squared error	0.0382	
Relative absolute error	12.2686 %	
Root relative squared error	36.5636 %	
Total Number of Instances	8776	

NB

Correctly Classified Instances	7174	81.7457 %
Incorrectly Classified Instances	1602	18.2543 %
Kappa statistic	0.6912	
Mean absolute error	0.0257	
Root mean squared error	0.1128	
Relative absolute error	116.9734 %	
Root relative squared error	108.019 %	
Total Number of Instances	8776	

RF

Correctly Classified Instances	7342	83.66 %
Incorrectly Classified Instances	1434	16.34 %
Kappa statistic	0.7424	
Mean absolute error	0.0072	
Root mean squared error	0.0648	
Relative absolute error	32.643 %	
Root relative squared error	61.9992 %	
Total Number of Instances	8776	

SMO

=== Summary ===

Correctly Classified Instances	7176	81.7685 %
Incorrectly Classified Instances	1600	18.2315 %
Kappa statistic	0.6915	
Mean absolute error	0.0064	
Root mean squared error	0.0777	
Relative absolute error	29.3112 %	
Root relative squared error	74.4105 %	
Total Number of Instances	8776	

Figure 20: Results of the J48 classifier, naïve Bayes classifier, random forests and SMO function with two-grams

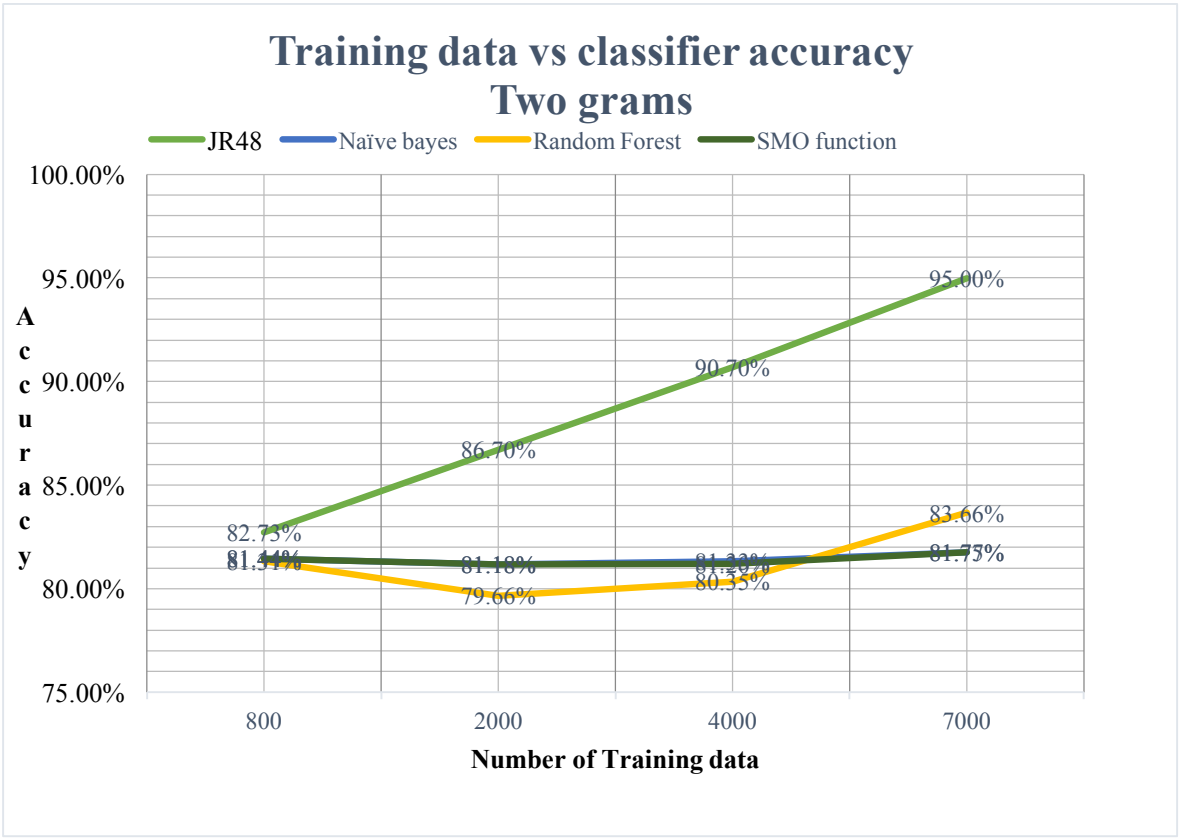


Figure 21: Two gram's classifier accuracy with different volumes of packets

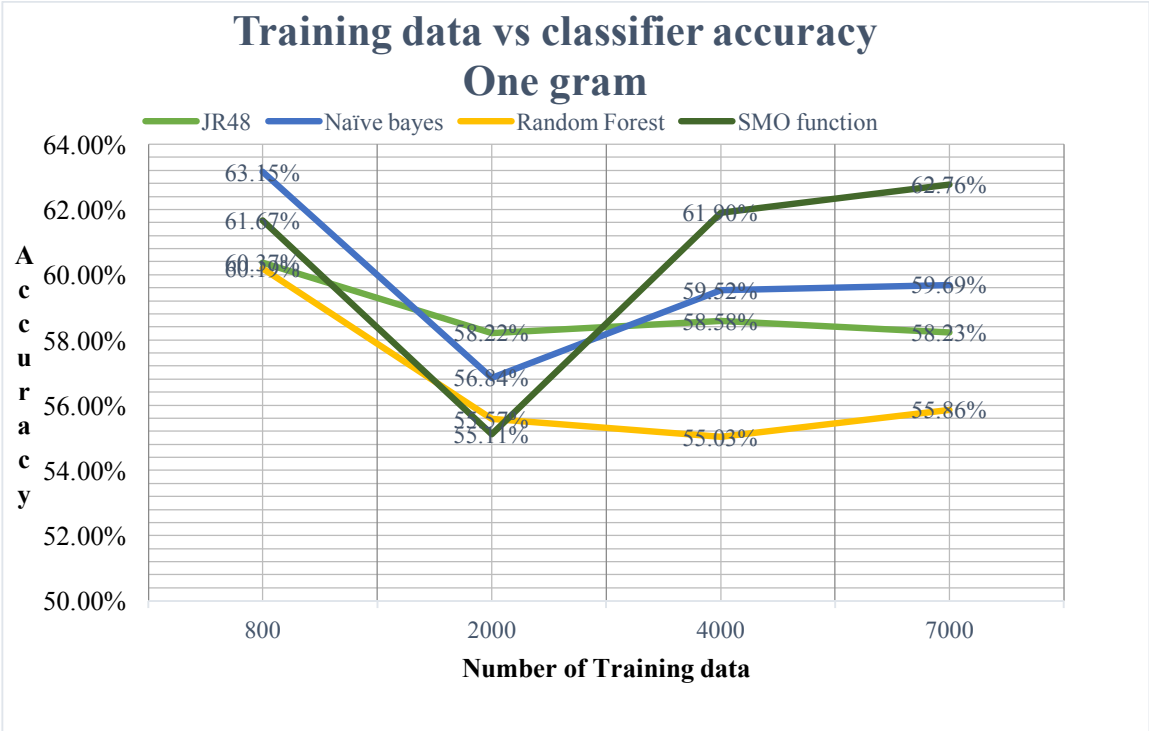


Figure 22: One gram's classifier accuracy with different volumes of packets

Chapter 5: Conclusion

We have introduced a novel and intelligent approach for faster packet matching by IDSs. In this approach, a classifier is first trained to predict the matching rule for any given packet and is then employed in the real network. This results in faster speed in terms of packet matching compared to a standard IDS (which browses through the set of rules to find the matching rule for a packet). We have proved the effectiveness of our approach both theoretically and empirically with real network traffic. We have also analyzed the different parameters of the system, such as the number of rules and the size of training data.

In the future, we would like to enhance intelligent predictive matching by introducing more efficient and sophisticated classifiers to address the issue of lower accuracy in the presence of a large number of rules. Besides this, we hope to apply different classifiers, big network data, and different attack scenarios to evaluate the robustness of our approach.

Bibliography

Acharya, S. a. M. B. N. a. A. M. a. Z. T. a. W. J. a. G. Z. a. G. A. G., 2007. OPTWALL: A Hierarchical Traffic-Aware Firewall.. In: *NDSS.* s.l.:s.n.

Andrés Felipe Arboleda, C. E. B., 2005. *~cbedon/snort/snortdevdiagrams.pdf.* [Online]
Available at: http://artemisa.unicauca.edu.co/~cbedon/snort/snortdevdiagrams.pdf
[Accessed 15 06 2016].

Baboescu, F. a. V. G., 2001. Scalable packet classification. *ACM SIGCOMM Computer Communication Review,* Volume 21, pp. 199--210.

berkeley, n.d. *intrusion-detection-guideline.* [Online]
Available at: https://security.berkeley.edu/intrusion-detection-guideline

bibing.us.es, n.d. *memoria%252Fpor_capitulos%252F04.snort.pdf.* [Online]
Available at:
http://bibing.us.es/proyectos/abreproy/12077/fichero/memoria%252Fpor_capitulos%252F04.snort.pdf
[Accessed 03 05 2016].

caida dataset, n.d. *passive 2012 dataset.xml.* [Online]
Available at: http://www.caida.org/data/passive/passive 2012 dataset.xml
[Accessed 15 03 2016].

Caswell, B. a. B. J. a. B. A., 2007. *Snort Intrusion Detection and Prevention Toolkit.* s.l.:Syngress.

Cohen, E. a. L. C., 2005. Packet classification in large ISPs: Design and evaluation of decision tree classifiers. In: *ACM SIGMETRICS Performance Evaluation Review.* s.l.:ACM, pp. 73--84.

dell, n.d. *Naive-Bayes-Classifier.* [Online]
Available at: http://documents.software.dell.com/Statistics/Textbook/Naive-Bayes-Classifier
[Accessed 06 11 2015].

El-Atawy, A. a. A.-S. E. a. T. T. a. B. R., 2009. Adaptive early packet filtering for defending firewalls against DoS attacks. In: *Infocom 2009, Ieee.* s.l.:IEEE, pp. 2437--2445.

El-Atawy, A. a. S. T. a. A.-S. E. a. L. H., 2007. Using online traffic statistical matching for optimizing packet filtering performance. In: s.l.:IEEE, pp. 866--874.

ferryas.lecturer.pens.ac.id, n.d. *SnortIneerWorking.pdf.* [Online]
Available at: http://ferryas.lecturer.pens.ac.id/NetSa_Papers/SnortIneerWorking.pdf
[Accessed 15 03 2016].

Fisk, M. a. V. G., 2002. *Applying fast string matching to intrusion detection,* s.l.: DTIC Document.

google, n.d. *HIDS - Events Detected.* [Online]
Available at: https://sites.google.com/site/idpsinfo498/home/host-based-intrusion-detection-systems---hids/hids---events-detected

Goyal, R. a. B. K. a. B. S., 2015. Packet classification. *GOOGLE.*

Gupta, P. a. M. N., 2001. Algorithms for packet classification. *IEEE,* Volume 15, pp. 24--32.

Hamed, H. a. E.-A. A. a. A.-S. E., 2006. On dynamic optimization of packet matching in high-speed firewalls. *Selected Areas in Communications, IEEE Journal on,* Volume 24, pp. 1817--1830.

Kencl, L. a. S. C., 2006. Traffic-adaptive packet filtering of denial of service attacks. In: *Proceedings of the 2006 International Symposium on on World of Wireless, Mobile and Multimedia Networks.* s.l.:IEEE Computer Society, pp. 485--489.

marc.info, 20. *snort-users.* [Online]
Available at: http://marc.info/?l=snort-users&m=130588413913432

Martin Roesch, C. G. S. C., 2014. *snort_manual.pdf.* [Online]
Available at: https://s3.amazonaws.com/snort-org/www/assets/166/snort_manual.pdf
[Accessed 6 11 2015].

mok, 2014. *state-transition-table-for-aho-corasick-algorithm.* [Online]
Available at: http://stackoverflow.com/questions/22398190/state-transition-table-for-aho-corasick-algorithm
[Accessed 25 03 2016].

Mothersole, I. a. R. M. J., 2011. Optimising rule order for a packet filtering firewall. In: *Network and Information Systems Security (SAR-SSI), 2011 Conference on.* s.l.:IEEE, pp. 1--6.

Murphy, C. a. S. D., 2012. An Analysis of the Snort Data Acquisition Modules. *SANS Institute InfoSec Reading Room,* Volume 34027.

Mustafa, U. a. M. M. M. a. T. Z. a. W. T. a. A. H. Z., 2013. Firewall performance optimization using data mining techniques. *Wireless Communications and Mobile Computing Conference (IWCMC), 2013 9th International,* pp. 934--940.

openmaniak.com, 2007. *inline_final.* [Online]
Available at: http://openmaniak.com/inline_final
[Accessed 6 11 2015].

sans, 2005. *host vs network based intrusion detection systems.* [Online]
Available at: https://cyber-defense.sans.org/resources/papers/gsec/host-vs-network-based-intrusion-detection-systems-102574
[Accessed 2015 11 06].

snort, n.d. *node31.html.* [Online]
Available at:
http://manual.snort.org/node31.html#SECTION00441000000000000000
[Accessed 06 11 2015].

Trabelsi, Z. a. Z. L. a. Z. S., 2011. Packet flow histograms to improve firewall efficiency. In: *Information, Communications and Signal Processing (ICICS) 2011 8th International Conference on.* s.l.:IEEE, pp. 1--5.

Trabelsi, Z. a. Z. S., 2012. Multilevel early packet filtering technique based on traffic statistics and splay trees for firewall performance improvement. In: *Communications (ICC), 2012 IEEE International Conference on.* s.l.:IEEE, pp. 1074--1078.

waikato University, n.d. *weka.* [Online]
Available at: http://www.cs.waikato.ac.nz/ml/weka/)
[Accessed 15 03 2016].

Woo, T. Y., 2000. A modular approach to packet classification: Algorithms and results. In: *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE.* s.l.:IEEE, pp. 1213--1222.

Appendix

One-gram feature vectors

arrOneGram.java

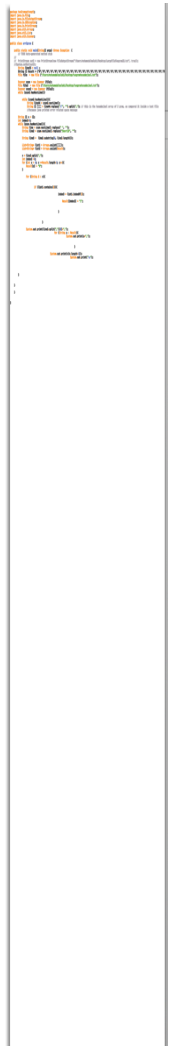
```java
package twoGramsattempt1;

import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.PrintStream;
import java.util.Arrays;
import java.util.List;
import java.util.Scanner;

public class arrOneGram {

    public static void main(String[] args) throws Exception  {
        // TODO Auto-generated method stub
        String line55 = null ;
        String [] Result = {"0","0","0","0","0","0","0","0","0","0","0","0","0","0","0","0","0","0","0","0","0","0","0","0","0","0","0","0","0","0","0","0"
        File file  = new File ("/Users/mohamedalmaleki/Desktop/oneGramLargeFile.txt");

        Scanner scan = new Scanner (file);
        File file2  = new File ("/Users/mohamedalmaleki/Desktop/oneGramLargeFile.txt");
        Scanner scan2 = new Scanner (file2);
        while (scan2.hasNextLine())

            while (scan2.hasNextLine()){
                 String line44 = scan2.nextLine();
                 String [] arr4 ={"00", "01", "02", "03", "04", "05", "06", "07", "08", "09", "0A", "0B", "0C", "0D", "0E", "0F","10","11", "12", "13","14"

        String [] v = {};
        int index2=0;
        while (scan.hasNextLine()){
            String line = scan.nextLine();
        //  String line = scan.nextLine().replace(" ", "");
        //  String line2 = scan.nextLine().replace("SnortID", "");

    //      String line3 =  line2.substring(0, line2.length());

            List<String> list1 = Arrays.asList(arr4);
            List<String> list2 = Arrays.asList(Result);

            v = line.split(",");
            int index3 =0;
            for (int z = 0; z <=Result.length-1; z++){
                Result[z] = "0";
            }
                for (String Q : v){

                        if (list1.contains(Q)){

                                        index3 = list1.indexOf(Q);
```

```
        //  String line2 = scan.nextLine().replace("SnortID", "");

//      String line3 =  line2.substring(0, line2.length());

        List<String> list1 = Arrays.asList(arr4);
        List<String> list2 = Arrays.asList(Result);

        v = line.split(",");
        int index3 =0;
        for (int z = 0; z <=Result.length-1; z++){
            Result[z] = "0";
        }
            for (String Q : v){

                    if (list1.contains(Q)){

                                        index3 = list1.indexOf(Q);

                                            Result[index3] = "1";

                                        }
                            }

            System.out.print(line.split(",")[0]+",");
                                    for (String u : Result){
                                                System.out.print(u+",");

                                                    }

                                System.out.print(v[v.length-1]);
                                            System.out.print("\n");


                    //=============================================================================


    }

}

}
}
```

Two-gram features vectors

```
String line55 = null ;
String [] Result = {"0","0","0","0","0","0","0","0","0","0","0","0","0","0","0","0","0","0","0","0","0","0","0","0","0","0","0","0","0","0","0","(
File file  = new File ("/Users/mohamedalmaleki/Desktop/twogramhexadecimal.txt");
String [] arr4 = {"B4B4","2800","0001","0100","0004","0400","0000","C2C5","C5CD","CDC4","C4FD","FDF9","F9FF","FF86","86E4","E49A","9AF8","F8FF","F
Scanner scan = new Scanner (file);
File file2  = new File ("/Users/mohamedalmaleki/Desktop/twogramhexadecimal.txt");
Scanner scan2 = new Scanner (file2);
while (scan2.hasNextLine())

    while (scan2.hasNextLine()){
        String line44 = scan2.nextLine();
//      String [] arr4 = line44.replace("\"", "").split(",");

String [] v = {};
int index2=0;
while (scan.hasNextLine()){
    String line = scan.nextLine().replace(" ", "");
    String line2 = scan.nextLine().replace("SnortID", "");

    String line3 =  line2.substring(0, line2.length());

    List<String> list1 = Arrays.asList(arr4);
    List<String> list2 = Arrays.asList(Result);

    v = line3.split(",");
    int index3 =0;
    for (int z = 0; z <=Result.length-1; z++){
        Result[z] = "0";
    }
```

```
38
39          String line3 =  line2.substring(0, line2.length());
40
41          List<String> list1 = Arrays.asList(arr4);
42          List<String> list2 = Arrays.asList(Result);
43
44          v = line3.split(",");
45          int index3 =0;
46          for (int z = 0; z <=Result.length-1; z++){
47              Result[z] = "0";
48          }
49
50              for (String Q : v){
51
52
53                      if (list1.contains(Q)){
54
55                                              index3 = list1.indexOf(Q);
56
57                                                  Result[index3] = "1";
58
59
60                                              }
61
62
63                                  }
64
65              System.out.print(line3.split(",")[0]+",");
66                                          for (String u : Result){
67                                                      System.out.print(u+",");
68
69
70                                                          }
71
72                                      System.out.print(v[v.length-1]);
73                                                      System.out.print("\n");
74
75
76
77
```



Two-gram source code

```
public static void main(String[] args) throws FileNotFoundException {

//  PrintStream out3 = new PrintStream(new FileOutputStream("/Users/mohamedalmaleki/Desktop/2gram.txt", true));
    //System.setOut(out3);
    // TODO Auto-generated method stub
    File noHexContent2 = new File("/Users/mohamedalmaleki/Desktop/Thesis/1.txt");
    Scanner line = new Scanner(noHexContent2);
    while (line.hasNextLine()){

        String len = line.nextLine();
        //String[] split = len.split(",");
        //String test = split[1].toString();

        for (int x =0; x<=len.length(); x++){
            int y=0;
            int z;
            //System.out.println("\n"+split[1]);

            while (y<=len.length()){

                System.out.print(len.substring(y, y+4) + ",   ");

                y = y+2;
                if (y+3 >= len.length()){

                    break;

                }
            }

            break;

        }
```

One-gram source code

```
public static void main(String[] args) throws Exception  {
    // TODO Auto-generated method stub

    String line55 = null ;
    String [] Result = {"0","0","0","0","0","0","0","0","0","0","0","0","0","0","0","0","0","0","0","0","0","0","0","0","0","0","0","0","0","0","0","0
//  File file  = new File ("/Users/mohamedalmaleki/Desktop/twogramhexadecimal.txt");

//  Scanner scan = new Scanner (file);
    File file2  = new File ("/Users/mohamedalmaleki/Desktop/twogramhexadecimal.txt");
    Scanner scan2 = new Scanner (file2);
    while (scan2.hasNextLine())

        while (scan2.hasNextLine()){
            String line44 = scan2.nextLine();
            String [] arr4 = line44.replace("\"", "").split(",");

    String [] v = {};
    int index2=0;
    while (scan.hasNextLine()){
        String line = scan.nextLine().replace(" ", "");
        String line2 = scan.nextLine().replace("SnortID", "");

        String line3 =  line2.substring(0, line2.length());

        List<String> list1 = Arrays.asList(arr4);
        List<String> list2 = Arrays.asList(Result);

        v = line3.split(",");
        int index3 =0;
        for (int z = 0; z <=Result.length-1; z++){
            Result[z] = "0";
        }

            for (String Q : v){


                if (list1.contains(Q)){

                            index3 = list1.indexOf(Q);

                                Result[index3] = "1";
```

```
                                                index3 = list1.indexOf(0);
                                                    Result[index3] = "1";
                                                }
                        }
                System.out.print(line3.split(",")[0]+",");
                                        for (String u : Result){
                                                System.out.print(u+",");
                                                    }
                                    System.out.print(v[v.length-1]);
                                                System.out.print("\n");
                        //=============================================================================
    }
}
}
```