A NOVEL SIGNATURE-BASED TRAFFIC CLASSIFICATION ENGINE TO REDUCE FALSE ALARMS IN INTRUSION DETECTION SYSTEMS

Md. Azizul Islam[1] and Md. Manirul Islam[2]

[1]Department of Computer Science, American International University-Bangladesh, Dhaka, Bangladesh.
[2]Department of Computer Science, American International University-Bangladesh, Dhaka, Bangladesh.

ABSTRACT

Pattern matching plays a significant role in ascertaining network attacks and the foremost prerequisite for a trusted intrusion detection system (IDS) is accurate pattern matching. During the pattern matching process packets are scanned against a pre-defined rule sets. After getting scanned, the packets are marked as alert or benign by the detection system. Sometimes the detection system generates false alarms i.e., good traffic being identified as bad traffic. The ratio of generating the false positives varies from the performance of the detection engines used to scan incoming packets. Intrusion detection systems use to deploy algorithmic procedures to reduce false positives though producing a good number of false alarms. As the necessities, we have been working on the optimization of the algorithms and procedures so that false positives can be reduced to a great extent. As an effort we have proposed a signature-based traffic classification technique that can categorize the incoming packets based on the traffic characteristics and behaviour which would eventually reduce the rate of false alarms.

KEYWORDS

Pattern Matching, False Positives, False Negatives, Signatures, NIDS

1. INTRODUCTION

In consequence of fast acceleration of security threats in today's computer network, Network Intrusion Detection Systems (NIDS) have become the critical components of contemporary network infrastructure. In a breathing network, while a NIDS is in traffic sniffing mode, it tends to execute a consecutive set of operations on receiving a packet like buffering the packet, matching the packet against signature set, and logging packets or alerts. Packet processing applications that have been recurrently used in our network today like Intrusion Detection/Prevention systems, Application layer switches, Packet filtering and transformation systems perform deep packet inspection while looking for predefined patterns in the packet payload.

Pattern matching aspects [32] for a fixed sequence of bytes in a single packet. Patterns are usually associated with particular services and source or destination ports in order to filter traffic inspection. For instance pattern matching can be notified as firing an alarm if a packet is IPv4 and UDP, having destination port 11570, and containing the string "bureaucracy" in the payload. Nevertheless, voluminous protocols and attacks don't make use of well-known ports, and consequently pattern matching has difficulty identifying these kinds of attacks. Furthermore, an enormous number of false positives can result if the matching is based on a pattern that is not so unique.

In order to define an attack Intrusion Detection Systems rely on pattern matching by comparing packets with known attack patterns. For a trusted intrusion detection system accurate pattern matching is the first requirement. As new attack patterns are being created frequently, since adaptive pattern matching is the second requirement. During the runtime frequently reconfiguration, fault can be effortlessly introduced either accidently or maliciously. Since faithful pattern reconfiguration is the third prerequisite to achieve trusted IDSs. Considering the requirements a trusted pattern matching engine [1] has been developed. This trusted pattern matching engine improves the detection accuracy by employing hybrid pattern matching engines: FPGA-based and multicore-based pattern matching engine. Both the engines work in parallel to scan incoming packets. The combine results of these two engines are used to determine the packet type and engine integrity. But the system still generates a good number of false alarms. In this research we have analysed a pattern matching engine architecture shown in figure 2 where an optimization of the algorithms and procedures have taken place to reduce the generation of false alarms.

Primarily we have focused on the applications [4] that make up the majority of traffic such as p2p, web folder and messenger applications that are used to exchange large amount of data. Figure 2 depicts a traffic classification engine architecture which deploys a packet classification engine in order to categorize the incoming packets by analysing the application specific traffic characteristics and behaviour. The classification engine generates application specific traffic queues. The queues get in line in the classified traffic queue module. The splitter then pulls out and forwards the packets from the traffic queue module to application specific detection engines. The detection engines then perform the pattern matching process with application specific traffic against a database of known attack patterns to determine the packet type (alert or benign). In my architecture the traffic classification engine classifies the incoming packets based on the traffic characteristics and behaviour. In order to evaluate the performance of this classification engine, the experiments with it have been implemented in an open source Network Intrusion Detection System- Snort.

2. RELATED WORKS

Ning Weng [1] proposed a trusted intrusion detection system shown in figure 1 by utilizing hybrid pattern matching engines: FPGA-based and multicore-based pattern matching engine.

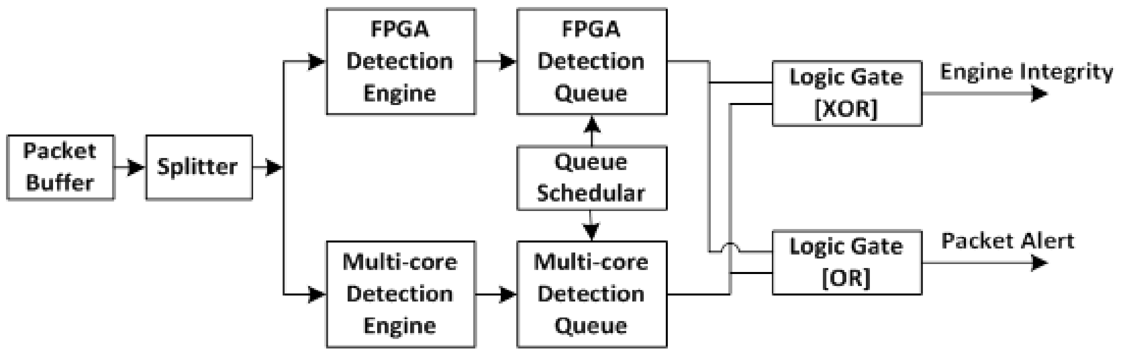


Figure 1. Trusted pattern matching engine

This trusted pattern matching engine improves the detection accuracy by employing two different detection engines. Both the engines work in parallel to scan the incoming packets. The combine results of these two engines are used to determine the packet type and engine integrity. But the system still generates a good number of false alarms.

A novel configurable string matching architecture have been designed by Lin Tan [13] that can store the all-inclusive Snort rule set in only 0.4 MB. It can run at upwards of 10 Gbps per instance. Lin Tan [14] developed a new string matching algorithm that was nearly 12 times more efficient than the currently best known approaches. Benjamin [15] presented and evaluated the architecture for high-throughput pattern matching of regular expressions. The limitations of current knowledge-based strategy to detect network attacks in a gradually complex and ever growing Internet are discussed by Casas [16]. In order to improve the performance of packet classification of Snort, Tongaonkar [17] presented a technique based on generating native code from Snort signatures.

Data-mining based approaches for network intrusion detection have been proposed by Priyanka V. [19]. The experimental results show that the suggested approach provides better performance in terms of accuracy and cost than the 'Knowledge Development and Data mining' (KDD) '99 cup [31] challenge. Bei Qi [20] proposed a new model of intrusion detection based on the data warehouse and the data mining. Soft Computing technique like Self organizing map for identifying intrusions in Network Intrusion Detection Systems have been suggested by Ranjani [9].

Xiaorong Cheng [7] deliberated a hybrid intrusion detection system based on Principle Component Analysis and Self organizing maps. This IDS aims at forming an extensible real-time intrusion model with high detection accuracy. Experimental results on the KDD 1999 Cup [31] dataset show that the proposed model is promising in terms of detection accuracy and computational efficiency, thus yielding for real-time intrusion detection. Antonatos [3] proposed Piranha, an algorithm for pattern matching designed specifically for intrusion detection. The experiments with Piranha implemented in a Network Intrusion Detection System (NIDS) - Snort v2.2 indicated that in terms of processing time, Piranha is faster than the existing algorithms up to 28%, and requires up to 73% less memory.

Intrusion Detection techniques [5] have been categorized as Misuse Detection, Anomaly Detection, and Specification-based approach. Anomaly detection systems are expected to detect novel attacks. Misuse detection systems distinguish known attack patterns. But such systems cannot detect novel attacks. As the necessities, Simon T. Powers [6] presented a hybrid system with the aim of combining the advantages of both approaches.

3. DOMAIN ANALYSIS

There are two primary approaches [21] to NIDS implementation-Signature based and Anomaly detection based. Signature based NIDSs retain a collection of signatures, each of which typifies the profile of a known security threat for instance a DoS attack or a virus. The signatures are used to analyse the data streams of numerous flows passing through the network link. When a flow matches a signature, apposite action is taken for instance block the flow or rate limit it.

Signature based systems are reactive as they combat against known attacks that have already affected and impaired a number of systems before being acknowledged. Conversely anomaly based systems are pre-emptive and autonomous that can ensure security without any manual interference. Some well-known commercial NIDS include AXENT, Cisco, CyberSafe, ISS, and Shadow while the popular open source NIDS includes Snort.

Pattern matching requires deep packet assessment that implicates scanning every byte of the packet payload. Patterns have been conventionally identified as precise match strings. Formerly numerous high speed and efficient string matching algorithms have been recommended due to their wide adoption and importance. In order to perform high performance string matching, a preprocessed data-structure has been accustomed by standard string matching algorithms such as Aho-Corasick [22], Commentz-Walter [23] and Wu-Manber [24].

3.1. Strengths of NIDS

A NIDS can implement the following functions to enhance the security of a network:

- Security policy enforcement for instance to block all communication between certain sets of IP addresses and or ports in a given network.
- Signature based NIDS can detect known worms, viruses, and exploit known security holes with fairly high degree of accuracy.
- New attacks and atypical patterns can be acknowledged by anomaly based NIDS in a network traffic whose signatures are not yet engendered.

3.2. Limitations of NIDS

Existing NIDS largely comprise the following limitations:

- After a NIDS falsely raise a security threat alarm for innocuous traffic, there triggers a false positive alarm.
- When a NIDS fails to raise a security threat alarm for harmful traffic, there initiates a false negative alarm.
- Regular expressions signatures used by signature based NIDS make a significant performance bottleneck. Long signatures [28] required to reduce false positives further reduces the performance.
- One and all tend to encrypt their data before transmission. After the packet payloads are encoded, the existing signatures [27] will become absolutely unusable in ascertaining the anomalous and harmful traffic.
- More or less all NIDS systems entail a continual human supervision, which decelerates the detection and the associated actions.
- Signature based NIDS are not capable to detect new attacks whose signatures are not yet developed.

4. DESIGN ANALYSIS

In this paper we have proposed a pattern matching engine architecture shown in figure 2 where an optimization of the algorithms and procedures have taken place to reduce the generation of false alarms.

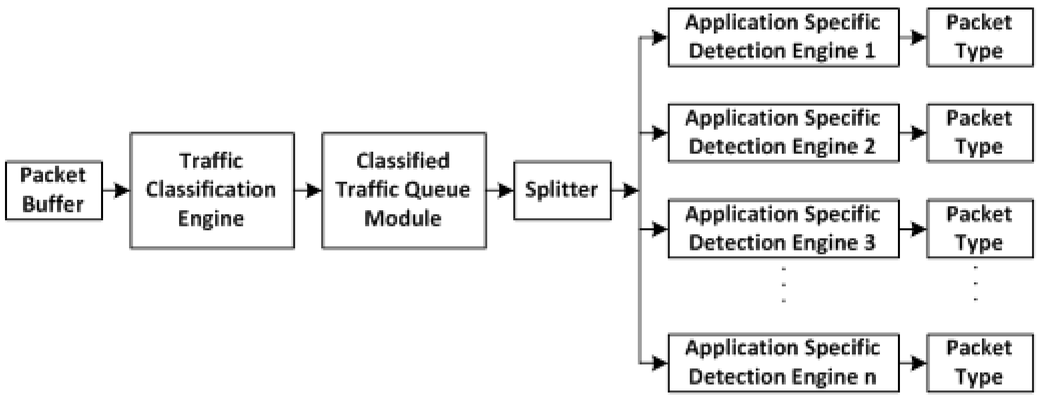

Packet Buffer
Traffic Classification Engine
Classified Traffic Queue Module
Splitter
Application Specific Detection Engine 1
Application Specific Detection Engine 2
Application Specific Detection Engine 3
Application Specific Detection Engine n
Packet Type
Packet Type
Packet Type
Packet Type

Figure 2. Pattern matching engine architecture

For traffic classification purpose, we have developed a packet sniffer application that can capture packets from online. Initially we have considered capturing HTTP, HTTPS and DNS traffic with ports 80, 443 and 53 respectively. The Pseudo-code for traffic classification engine is shown in figure 3. From the given pseudo-code it is seen that the traffic classification engine classifies the incoming packets based on the protocol field. After the classification gets done, traffic classification engine generates protocol specific in fact application specific traffic queues and forward them to classified traffic queue module. Unlike traditional intrusion detection systems, at this point the splitter pulls out and forwards the packets from the classified traffic queue module to application specific detection engines.

```
Receive the incoming packets in Packet Buffer
Segregate the traffic based on the protocol field
If Src/Dst port = 443 Then
    Capture traffic and forward to HTTPS traffic queue module
If Src/Dst port = 53 Then
    Capture traffic and forward to DNS traffic queue module
If Src/Dst port = 80 Then
    Capture traffic and forward to HTTP traffic queue module
Else
    Identify and Segregate new traffic
End If
```

Figure 3. Pseudo-code for traffic classification engine

Traditional intrusion detection systems perform the pattern matching process in a random manner like search for predefined patterns in the incoming packet payload against a database of known attack patterns. But in my architecture, as the detection engines are application specific, a particular detection engine will perform the pattern matching process for specific application only to determine the packet type. Literally it is clear that an application specific detection engine will perform better pattern matching than that of a generic detection engine. As application specific detection engines are dealing with a database of application specific known attack patterns-thus reducing the generation of false alarms while defining the packet type.

The sniffer has been developed with an external packet sniffing library: Libpcap. For experiment purpose we have used Lubuntu 12.10 which is a Debian based distribution. We have installed this operating system on VirtualBox.

At the very beginning of the deployment of my architecture we have given a closer look to the world’s most popular open source Network Intrusion Detection System (NIDS) Snort. In my architecture the traffic classification engine classifies the incoming packets based on the traffic characteristics and behaviour. The experiments with this classification engine have been implemented with Snort for performance evaluation. In order to determine the false positive rate with Snort, we have analysed the functionality of Snort; Snort’s unified log application- Barnyard2, Network security monitoring application- Snorby and the Database- MySQL.

4.1. Snort

Snort is an open source Intrusion Detection/Prevention system equipped with real time traffic analysis and packet logging features. Snort can be divided into five major components shown in figure 4 that are each critical to intrusion detection.

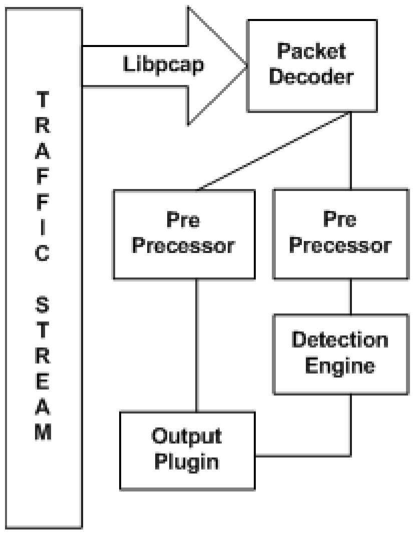


Figure 4. Snort component data-flow

4.1.1. Libpcap

Snort uses an external packet sniffing library-Libpcap. It is an open source library that provides a high level interface to network packet capture systems. The library is a platform independent API that can eliminate the need for system-dependent packet capture modules in each application. Libpcap [33] makes the capture facility for raw packets provided by the underlying operating system available to other applications. A raw packet has all its protocol header information left intact and inviolate by the operating system. Network applications usually do not process raw packets; they depend on the OS to read protocol information and appropriately frontward payload data to them. But Snort typically requires the opposite: it needs to have the packets in their raw state to function. Snort uses protocol header information that would have been stripped off by the operating system to detect some forms of attacks.

4.1.2. Packet Decoder

As soon as packets have been gathered, Snort starts deciphering [33] the specific protocol elements for each packet.

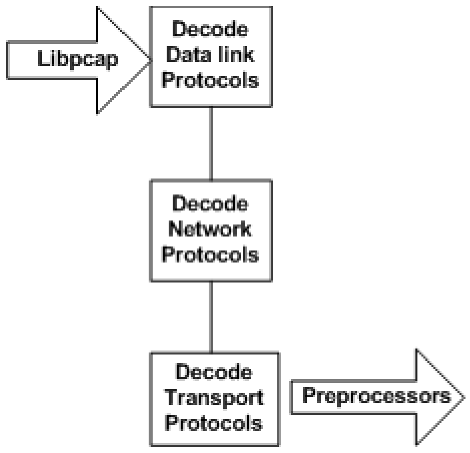


Figure 5. Packet decoder data-flow

Fundamentally the packet decoder is a series of decoders that each decodes specific protocol elements. It works up the Network stack, starting with lower level Data link protocols, decoding each protocol as it moves up. A packet follows this data flow as it moves through the packet decoder shown in figure 5. As packets move through the various protocol decoders, a data structure is filled up with decoded packet data. Once packet data is stored in a data structure it is ready to be analyzed by the preprocessors and the detection engine.

4.1.3. Preprocessor

Preprocessor parameters [33] are configured and tuned via the ‘snort.conf’ file. The ‘snort.conf’ file let us add or remove preprocessors as we see fit. Several preprocessors that come with a standard Snort distribution are listed below:

- *protocol decoders*: These preprocessors are used to normalize various TCP traffic streams including: telnet, ftp, smtp, and rpc.
- *http_inspect*: This preprocessor normalizes http traffic.
- *frag3 preprocessor*: Used to reassemble packet fragments prior to inspection.
- *stream5 preprocessor*: Used to reconstruct TCP data streams so that inspection can be done in the context of a TCP conversation rather than the individual packets that make up the stream.
- *DCE/RPC2*: Used to decode and desegment DCE traffic.
- *sfPortscan*: A portscan detection plug-in.

4.1.4. Detection Engine

The detection engine is the principal Snort component. It has the following key roles: Rules parsing and Signature detection. The detection engine [33] constructs attack signatures by analyzing Snort rules. Snort rules are read line by line, and are loaded into an internal data structure. Rules are alienated into two functional segments:

- The rule header (rule tree node)
- The rule option (option tree node)

The rule header encompasses data about the conditions for applying the signature. The rule header for a DoS rule is:

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS 80
```

The rule option for the same rule begins and ends with a parenthetical. The rule option comprises the concrete signature and certain credentials about the attack:

```
(msg:”DOS Cisco attempt”; flow:to_server,established; dsize:1; content:”|13|”; classtype:web-application-attack; sid:1545; rev:8;)
```

4.1.5. Output Plugins

The output module handles the task of writing and displaying events. Snort-generated data can be presented to standard output, logged to files or logged to a database. The output module is somewhat unique in that it can receive input from various sources including the following:

- The packet decoder to produce tcpdump-style output.
- Preprocessors can send alerts directly to the output module on detection of anomalous conditions as they inspect and assemble packets and packet streams.
- The detection engine sends log and alert data to the output module when rules are matched.

Snort has the following output plugins [33] that drive out data in different formats:

- Log_tcpdump

- Alert_unixsock
- Alert_full
- Alert_smb
- Alert_fast
- XML
- Alert_syslog
- CSV
- Database
- Unified2

4.1.6. Database

The database output plugin logs straight to a relational database. We have chosen the world's most popular open source database-MySQL in our experiment. The database plugin [33] has the following three options:

- *log or alert:* This option let us write alert or log data to the selected database. To write both alert and log data to the same database, the database plugin is required to be activated twofold, once with *alert* and once with *log*.
- *mysql or postgresql or unixodbc or mssql*: This is the type of database we are going to write to.
- *Parameter list*: The Parameter list comprises the *name=value* pairs required to positively write to a database. The names and descriptions are as follows:
    - host: The host is set to localhost as it is connected to a database local to snort.
    - port: The port number to connect at the remote database host.
    - dbname: The database username required to log in to a database.
    - password: The password prerequisite to database authentication.
    - sensor_name: Snort processes the sensor name in the database.
    - hex: This name is included with no value to store binary data in hex. This is the default.
    - base64: This name is included with no value to store binary data in base64 encoding.
    - ascii: This name is included with no value to store binary data in human-readable ASCII encoding.
    - detail: It can be either full or fast. Full logs identical specifics of the packet and Fast logs signature, timestamp, source and destination IPs and ports, TCP flags, and protocol.

4.1.7. Unified2

Unified2 writes intrusion data to its own binary format. The purpose of the unified2 plugin [33] is to allow data to be pushed out of Snort as fast as possible and to outsource plugins to a dedicated application. The application-Barnyard2 reads unified2 output and sends data to other plugins, namely database output plugins. Unlike other plugins, the unified2 plugin is enabled with two different commands. The commands are given below:

- alert_unified: This command outputs alert data.
- log_unified: This command outputs log data.

Both the commands have one configuration option:

- limit [maximum size]: Default file size is 128MB.

4.2. Barnyard2

Barnyard2 is an open source interpreter [26] for Snort unified2 binary output files. Barnyard2 has 3 modes of operation:

- *batch (or one-shot)*: In batch mode, barnyard2 will process the explicitly specified files and exit.
- *continual*: In continual mode, barnyard2 will start with a location to look and a specified file pattern and continue to process new data (and new spool files) as they appear.
- *continual w/ bookmark*: Continual mode w/ bookmarking will also use a checkpoint file (or waldo file in the snort world) to track where it is. In the event the barnyard2 process ends while a waldo file is in use, barnyard2 will resume processing at the last entry as listed in the waldo file.

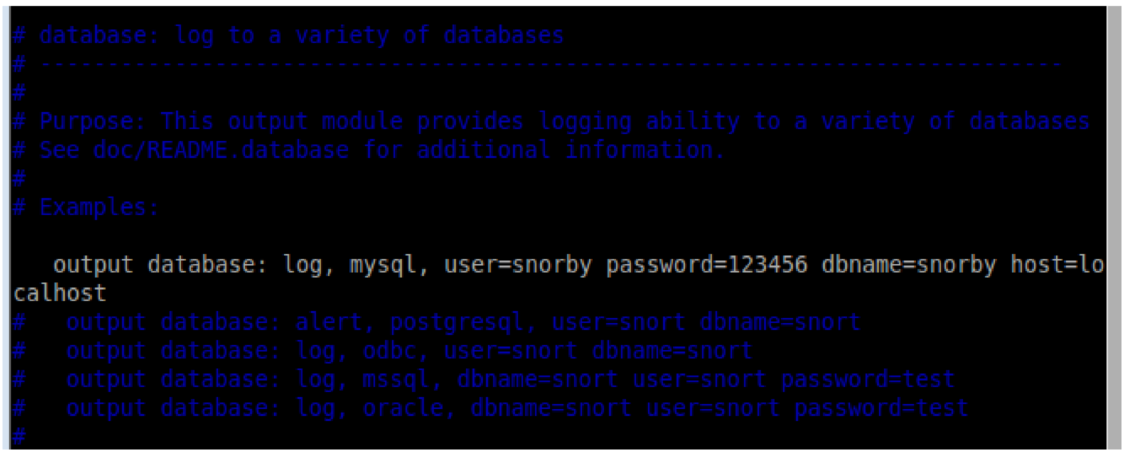
The "-f", "-w", and "-o" options are used to determine which mode barnyard2 will run in. It is legal for both the "-f" and "-w" options to be used on the command line at the same time, however any data that exists in the waldo file will override the command line data from the "-f" and "-d" options.

4.3. Snorby

Snorby is a ruby on rails web application [25] for network security monitoring that interfaces with Snort. Snorby fetches our existing and new network security monitoring data to life with a suite of beautiful, relevant and actionable metrics. With a simple keystroke or a mouse click, we can quickly classify an event into one of the many preconfigured classifications or into custom classifications relevant to our needs. Classifications can be used to unify events into helpful categories for follow-up investigations or for tuning our alert rule sets.

4.3.1. Barnyard2 Event Feed

Snorby gathers events from sensors. On our local server, we configured Barnyard2 to write Snort events to the Snorby database shown in figure 6.



```
# database: log to a variety of databases
# ----------------------------------------------------------------------------
#
# Purpose: This output module provides logging ability to a variety of databases
# See doc/README.database for additional information.
#
# Examples:

   output database: log, mysql, user=snorby password=123456 dbname=snorby host=lo
calhost
#   output database: alert, postgresql, user=snort dbname=snort
#   output database: log, odbc, user=snort dbname=snort
#   output database: log, mssql, dbname=snort user=snort password=test
#   output database: log, oracle, dbname=snort user=snort password=test
#
```

Figure 6. Barnyard2 event feed

After the configuration we restarted barnyard2 and the events started to pop up in the GUI shown in figure 7.

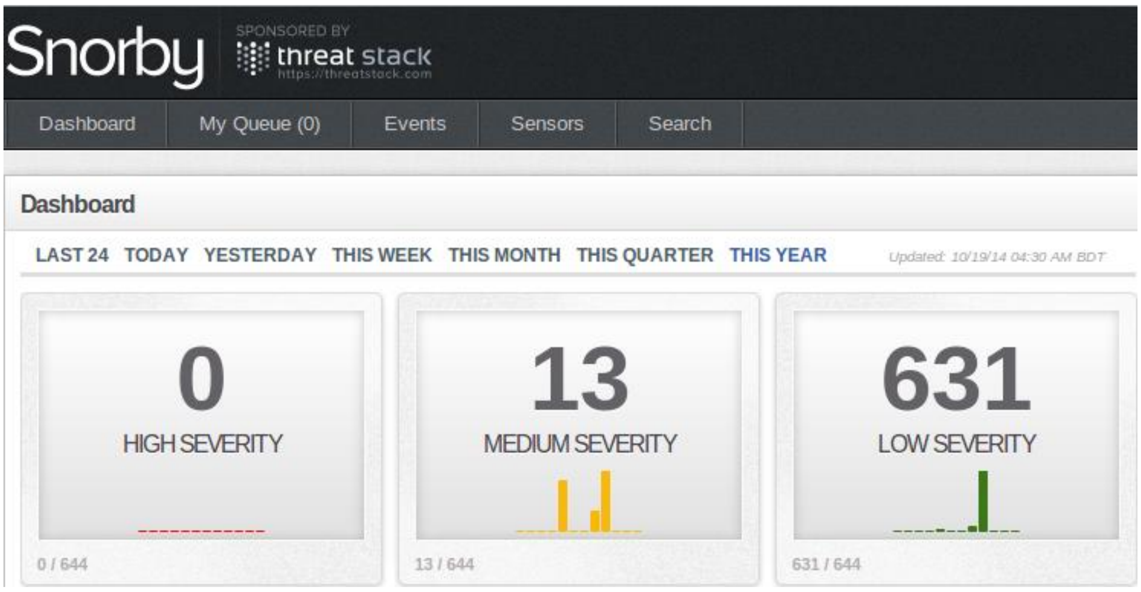


Figure 7. Snorby GUI

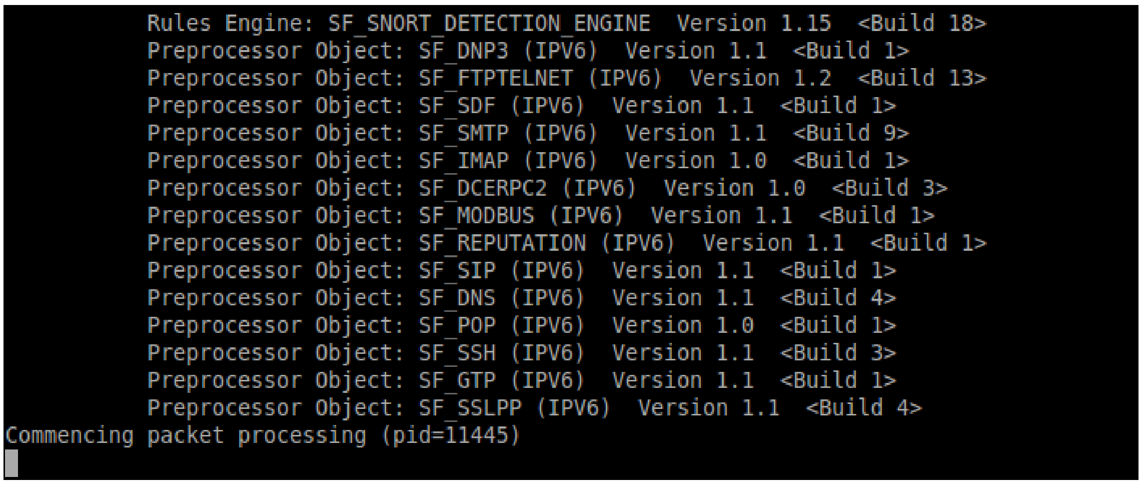
5. RESULT ANALYSIS

So far we have seen that in order to determine the false positive rate in Snort, first of all we need to run Snort in packet sniffing mode to capture packets online. Snort captures packets off the wire; scan the packets against a database of known attack patterns to determine the packet type; logs packets or alerts.

5.1. Packet Capture with Snort

Experimentally we opened the chromium web browser to various webpages having chat on facebook, and made video calls on skype. Concurrently Snort was executed to capture live data shown in figure 8 with the following command:

```
# snort - c /etc/snort/snort.conf - l /var/log/snort
```



```
           Rules Engine: SF_SNORT_DETECTION_ENGINE  Version 1.15  <Build 18>
           Preprocessor Object: SF_DNP3 (IPV6)  Version 1.1  <Build 1>
           Preprocessor Object: SF_FTPTELNET (IPV6)  Version 1.2  <Build 13>
           Preprocessor Object: SF_SDF (IPV6)  Version 1.1  <Build 1>
           Preprocessor Object: SF_SMTP (IPV6)  Version 1.1  <Build 9>
           Preprocessor Object: SF_IMAP (IPV6)  Version 1.0  <Build 1>
           Preprocessor Object: SF_DCERPC2 (IPV6)  Version 1.0  <Build 3>
           Preprocessor Object: SF_MODBUS (IPV6)  Version 1.1  <Build 1>
           Preprocessor Object: SF_REPUTATION (IPV6)  Version 1.1  <Build 1>
           Preprocessor Object: SF_SIP (IPV6)  Version 1.1  <Build 1>
           Preprocessor Object: SF_DNS (IPV6)  Version 1.1  <Build 4>
           Preprocessor Object: SF_POP (IPV6)  Version 1.0  <Build 1>
           Preprocessor Object: SF_SSH (IPV6)  Version 1.1  <Build 3>
           Preprocessor Object: SF_GTP (IPV6)  Version 1.1  <Build 1>
           Preprocessor Object: SF_SSLPP (IPV6)  Version 1.1  <Build 4>
Commencing packet processing (pid=11445)
```

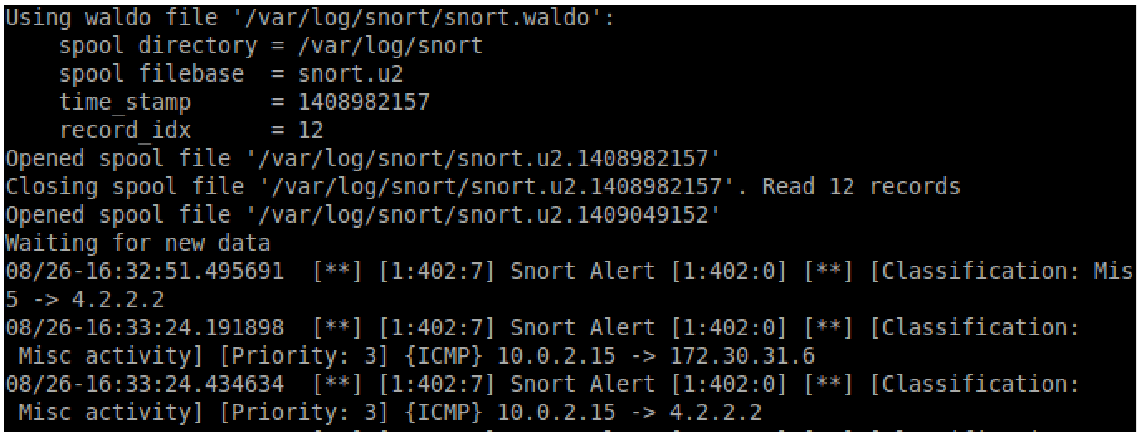
Figure 8. Snort packet Capture

As soon as the packets gathered, Snort started decoding the specific protocol elements for each packet. As packets moved through the various protocol decoders, a data structure was filled up with decoded packet data. After packet data was stored in a data structure it was ready to be analyzed by the preprocessors and the detection engine. Preprocessors were used to either examine packets for suspicious activity or modify packets so that the detection engine could properly interpret them. As a primary Snort component the detection engine built attack signatures by parsing Snort rules. Snort rules were read line by line, and were loaded into an internal data structure. The detection engine processed rule headers and rule options differently. The detection engine built a linked list decision tree. The nodes of the tree tested each incoming packet for increasingly precise signature elements. The process continued until the packet either matched an attack signature or tested clean or was dropped.

Unified2 is the output plugin designed for outputting Snort intrusion data. Unified2 wrote intrusion data to its own binary format. We executed Barnyard2 with the following command:

```
# barnyard2 - c /etc/snort/barnyard2.conf - d /var/log/snort - f snort.u2 - w
/var/log/snort/snort.waldo
```

Barnyard2 read unified2 logs shown in figure 9 and sent those logs to MySQL.



```
Using waldo file '/var/log/snort/snort.waldo':
    spool directory = /var/log/snort
    spool filebase  = snort.u2
    time_stamp      = 1408982157
    record_idx      = 12
Opened spool file '/var/log/snort/snort.u2.1408982157'
Closing spool file '/var/log/snort/snort.u2.1408982157'. Read 12 records
Opened spool file '/var/log/snort/snort.u2.1409049152'
Waiting for new data
08/26-16:32:51.495691  [**] [1:402:7] Snort Alert [1:402:0] [**] [Classification: Mis
5 -> 4.2.2.2
08/26-16:33:24.191898  [**] [1:402:7] Snort Alert [1:402:0] [**] [Classification:
 Misc activity] [Priority: 3] {ICMP} 10.0.2.15 -> 172.30.31.6
08/26-16:33:24.434634  [**] [1:402:7] Snort Alert [1:402:0] [**] [Classification:
 Misc activity] [Priority: 3] {ICMP} 10.0.2.15 -> 4.2.2.2
```

Figure 9. Barnyard2 Spooling Intrusion Data

Snorby [35] is a Ruby on Rails based frontend for Snort with the following features:

- Dashboard with Reporting:
  - Number of events by severity (high, medium, low)
  - event count vs time by sensor
  - severity count vs time
  - protocol count vs time
  - signature distribution graph
  - source distribution graph
  - destination distribution graph
- Events: timeline of events with details, including OpenFPC features
- Sensors: list of sensors
- Search: enables to filter events by criteria
- Administration: admin backend of the application

In our experiment, we captured all the incoming packets through the eth0 interface. In the Snorby dashboard we found a graph for the number of events count vs time shown in figure 10.

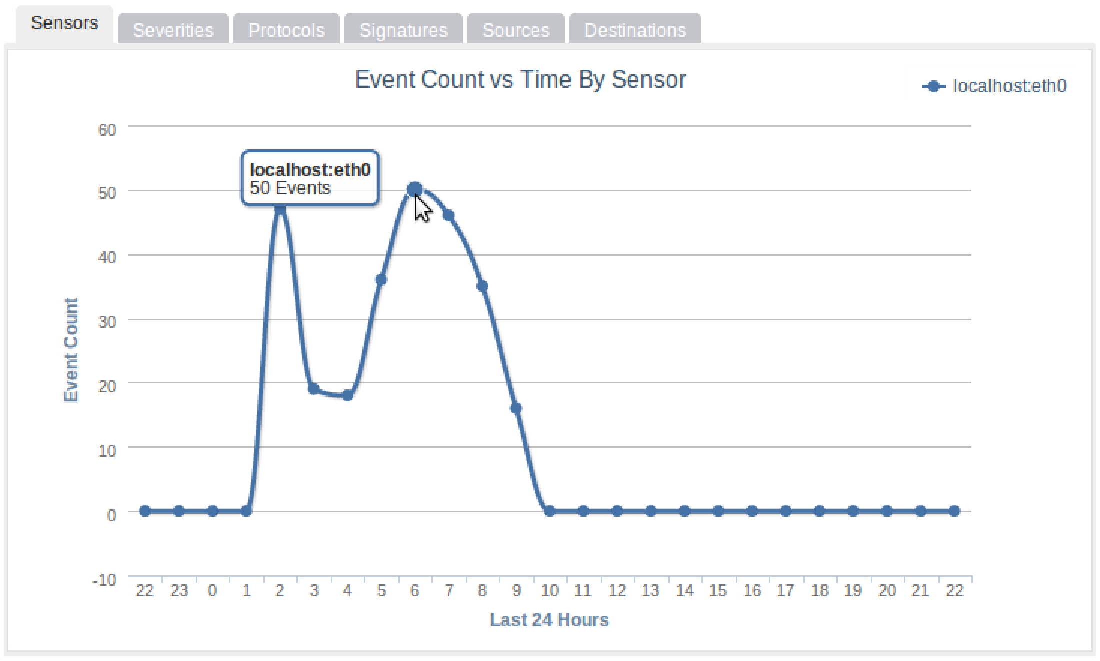


Figure 10. Event count vs time by sensor

Considering Severity Snorby categorized events shown in figure 11.

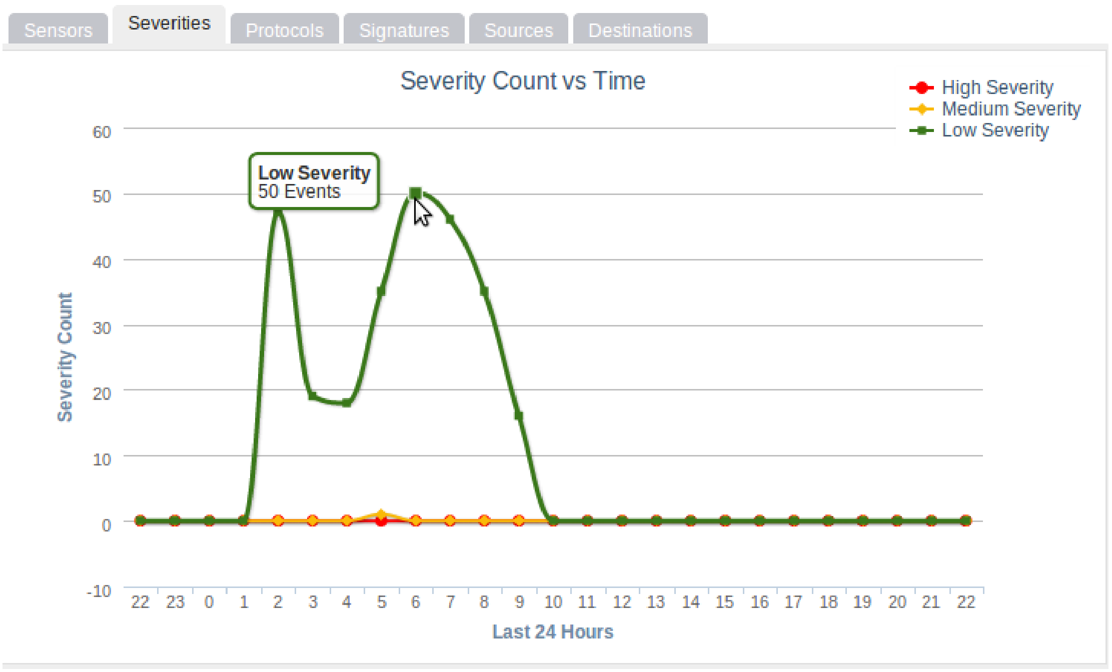


Figure 11. Severity count vs time

We can list out the event signatures that were used by the Snort detection engines while scanning the incoming packets. Beneath the signatures that have been used so far in this experiment:

- Snort Alert [1:402:0]
- Snort Alert [1:527:0]
- Snort Alert [1:254:0]

- Snort Alert [1:368:0]
- Snort Alert [1:366:0]
- Snort Alert [1:384:0]
- Snort Alert [1:408:0]

5.1.1. Event signature Snort Alert [1:402:0] information

In this signature the signature name is Snort Alert [1:402:0] where 402 is the sig_sid and 1 is the sig_gid. The sid keyword [34] is used to uniquely identify Snort rules. Generally sid's between 100 and 999,999 are included with the Snort distribution. And sid's that are greater than or equal to 1,000,000 used for local rules.

- Format

sid: <snort rules id>;

- Example

This example is rule with the Snort Rule ID of 273.

```
alert ip $EXTERNAL_NET any -> $HOME_NET any (msg:"DOS IGMP dos attack"; fragbits:M+; ip_proto:2; reference:bugtraq,514; reference:cve,1999-0918; classtype:attempted-dos; sid:273; rev:8;)
```

The rev keyword is used to uniquely identify revisions of Snort rules.

- Format

rev: <revision integer>;

- Example

This example is rule with the Snort Rule Revision of 8.

```
alert ip $EXTERNAL_NET any -> $HOME_NET any (msg:"DOS IGMP dos attack"; fragbits:M+; ip_proto:2; reference:bugtraq,514; reference:cve,1999-0918; classtype:attempted-dos; sid:273; rev:8;)
```

The gid keyword (generator id) is used to identify what part of Snort generates the event when a particular rule fires. To avoid potential conflict with gids defined in Snort, it is recommended that values starting at 1,000,000 be used.

- Format

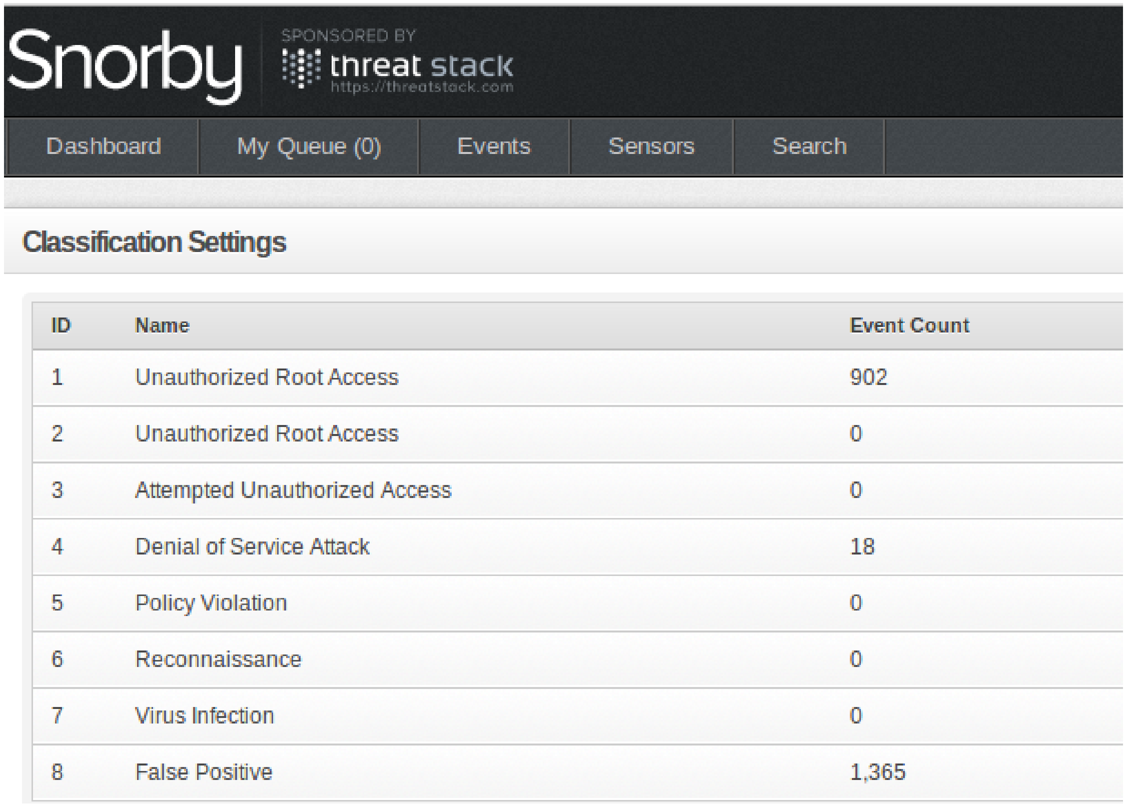
gid: <generator id>;

- Example

This example is a rule with a generator id of 1000773.

```
alert tcp any any -> any 80 (content:"web-attack"; gid:1000773; sid:1; rev:1;)
```

After running Snort for around 8 (Eight) hours, 2411 events were found by the eth0 sensor.
In order to organize events into helpful categories we used Snorby classification settings shown in figure 12.

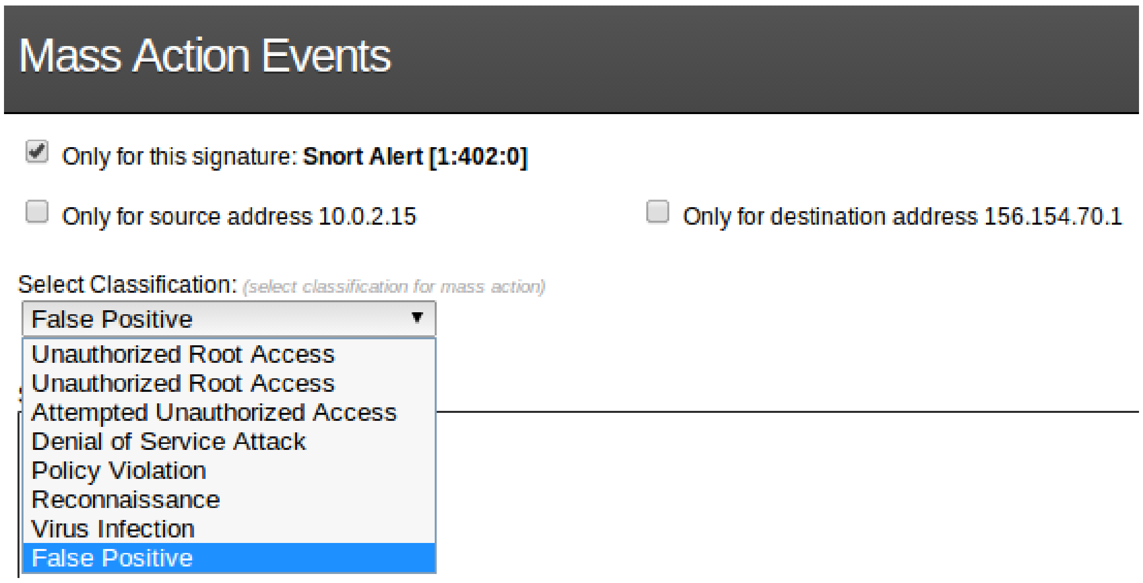


Classification Settings

ID	Name	Event Count
1	Unauthorized Root Access	902
2	Unauthorized Root Access	0
3	Attempted Unauthorized Access	0
4	Denial of Service Attack	18
5	Policy Violation	0
6	Reconnaissance	0
7	Virus Infection	0
8	False Positive	1,365

Figure 12. Snorby Event Categories

As I mentioned earlier that an event can be classified with a simple keystroke or a mouse click shown in figure 13.



[✓] Only for this signature: **Snort Alert [1:402:0]**

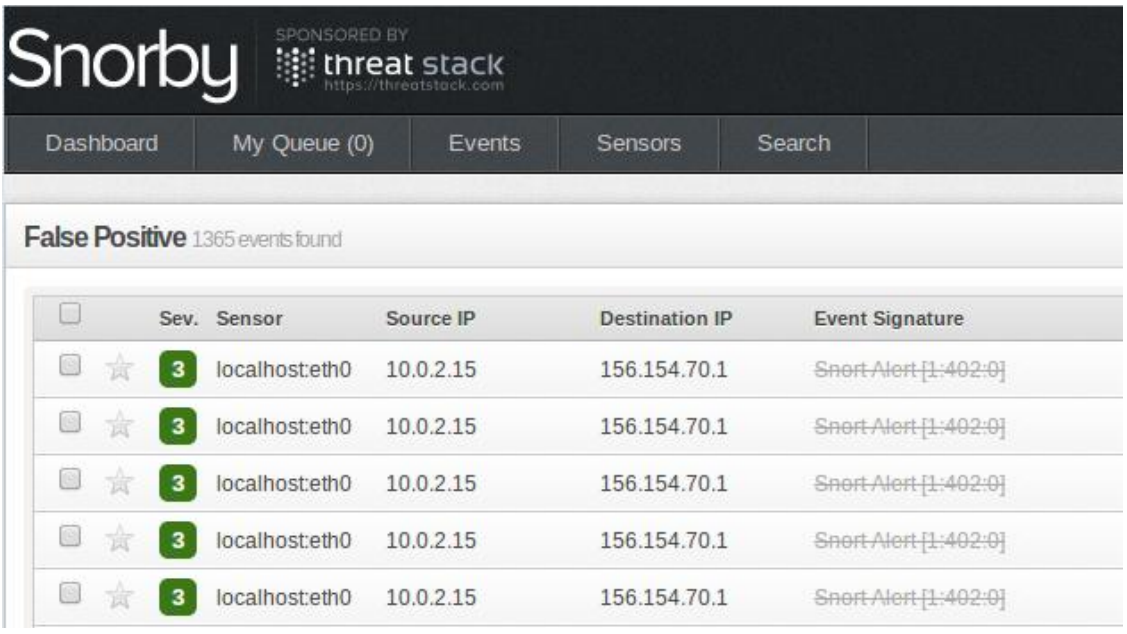
[ ] Only for source address 10.0.2.15 [ ] Only for destination address 156.154.70.1

Select Classification: *(select classification for mass action)*

False Positive ▼
- Unauthorized Root Access
- Unauthorized Root Access
- Attempted Unauthorized Access
- Denial of Service Attack
- Policy Violation
- Reconnaissance
- Virus Infection
- False Positive

Figure 13. Snorby Event Classification

After performing this classification we found a good number (1365) of false positive events, in percentage which is approximately 57% of the total events found by the eth0 sensor shown in figure 14.



False Positive 1365 events found

		Sev.	Sensor	Source IP	Destination IP	Event Signature
☐	☆	3	localhost:eth0	10.0.2.15	156.154.70.1	~~Snort Alert [1:402:0]~~
☐	☆	3	localhost:eth0	10.0.2.15	156.154.70.1	~~Snort Alert [1:402:0]~~
☐	☆	3	localhost:eth0	10.0.2.15	156.154.70.1	~~Snort Alert [1:402:0]~~
☐	☆	3	localhost:eth0	10.0.2.15	156.154.70.1	~~Snort Alert [1:402:0]~~
☐	☆	3	localhost:eth0	10.0.2.15	156.154.70.1	~~Snort Alert [1:402:0]~~

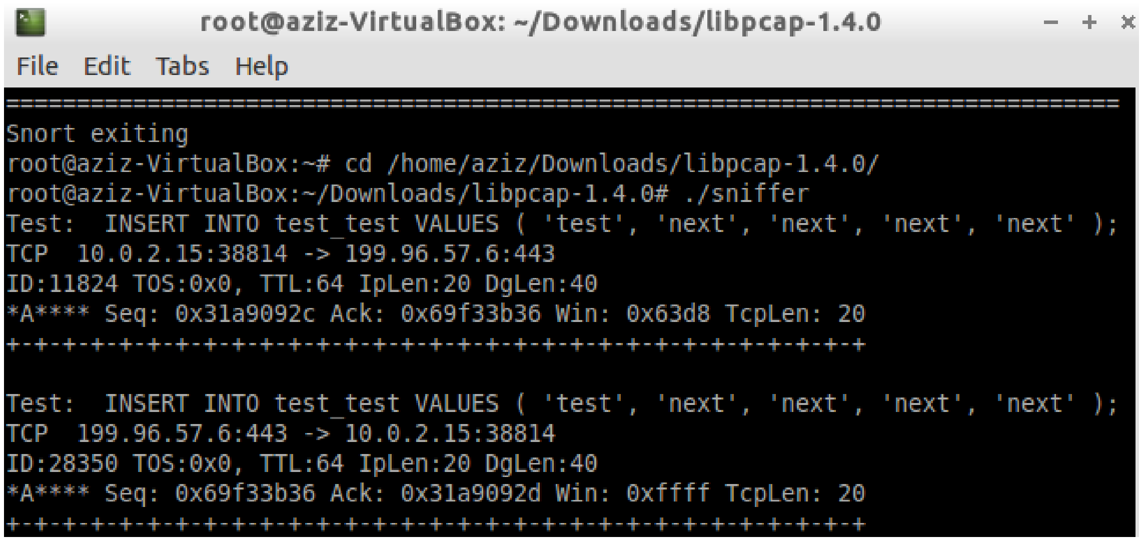
Figure 14. False Positive Events

5.2. Packet Capture with Sniffer

In order to reduce the generation of false alarms, we have developed a sniffer that can classify the incoming packets based on the protocol field just after capturing them. The sniffer captured packets with the external packet capturing library: Libpcap. The sniffer was executed to sniff packets off the wire shown in figure 15 with the following command.

```
#. /sniffer
```

The sniffer application was tested by getting the traffic between the local system and the Internet. The sniffer looked for HTTP, HTTPS and DNS traffic with source or destination ports 80, 443 and 53 respectively. We ran sniffer as root, then we opened a browser to various web pages. The output was something like the following. As we could see our browser client - 10.0.2.15 started the connection by going through the TCP handshaking by exchanging SYN and ACK packets.



```
===============================================================================
Snort exiting
root@aziz-VirtualBox:~# cd /home/aziz/Downloads/libpcap-1.4.0/
root@aziz-VirtualBox:~/Downloads/libpcap-1.4.0# ./sniffer
Test:  INSERT INTO test_test VALUES ( 'test', 'next', 'next', 'next', 'next' );
TCP  10.0.2.15:38814 -> 199.96.57.6:443
ID:11824 TOS:0x0, TTL:64 IpLen:20 DgLen:40
*A**** Seq: 0x31a9092c Ack: 0x69f33b36 Win: 0x63d8 TcpLen: 20
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

Test:  INSERT INTO test_test VALUES ( 'test', 'next', 'next', 'next', 'next' );
TCP  199.96.57.6:443 -> 10.0.2.15:38814
ID:28350 TOS:0x0, TTL:64 IpLen:20 DgLen:40
*A**** Seq: 0x69f33b36 Ack: 0x31a9092d Win: 0xffff TcpLen: 20
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 15. Sniffer Packet Capture

After executing the sniffer for a couple of minutes, it was found that the sniffer classified the incoming packets based on the protocol field. After the classification got done, traffic classification engine generated protocol specific in fact application specific traffic queues and forwarded them to classified traffic queue module. Unlike traditional intrusion detection systems, at this point the splitter pulled out and forwarded the packets from the classified traffic queue module to application specific detection engines. Traditional intrusion detection systems perform the pattern matching process in a random manner like search for predefined patterns in the incoming packet payload against a database of known attack patterns. But in my architecture, as the detection engines were application specific, a specific detection engine performed the pattern matching process for specific application only to determine the packet type.

Literally it is clear that an application specific detection engine will perform better pattern matching than that of a generic detection engine as application specific detection engines are dealing with a database of application specific known attack patterns-thus reducing the generation of false alarms while defining the packet type.

6. CONCLUSION

Throughout this research we have tried to make an analysis on the optimization of hybrid pattern matching so that false positives can be reduced to a great extent. On the way to reduce false positives, we have introduced a traffic classification engine in our architecture that can classify the incoming packets based on the protocol field just after capturing the packets. Classified traffic queues have been generated to be forwarded to application specific detection engines in order to reduce the generation of false alarms. So far we have seen that the detection engines are generic in determining the packet type. In our architecture the traffic classification engine classifies packets considering some specific protocols. But it is possible to intensify the functionality of the traffic classification engine where it would be possible to categorize the incoming packets considering more protocols. Moreover we have an area of on-going research in application specific detection engine design where the detection engines would be able to determine the packet type more efficiently against a database of predefined rule-sets.

REFERENCES

[1] B. Soewito, L. Vespa, N. Weng, H. Wang, “Hybrid Pattern Matching for Trusted Intrusion Detection”, Security and Communication Networks, Wiley Online Library, Volume 4, Issue 1, pages 33–43, January 2011.

[2] Sarang Dharmapurikar, John Lockwood, “Fast and Scalable Pattern Matching for Network Intrusion Detection Systems”, IEEE Journal on Selected Areas in Communication, Volume 24, Issue 10, pages 1781-1792, October 2006.

[3] S. Antonatos , M. Polychronakis , P. Akritidis , K. G. Anagnostakisy , E. P. Markatos, “Fast and Memory-Efficient Pattern Matching for Intrusion Detection”, In Proceedings 20th IFIP International Information Security Conference (SEC 2005).

[4] Yongmin Choi, KT Corp., Daejeon, “On the Accuracy of Signature Based Traffic Identification Technique in IP Networks”, 2nd IEEE/IFIP International Workshop on Broadband Convergence Networks, Issue 21-21, Pages 1-12, May 2007.

[5] G.D.Kurundkar, N.A.Naik, Dr. S.D. Khamitkar, Dr. N.V. Kalyankar, “Network Intrusion Detection by Applying Various Testing Techniques”, Global Journal of Computer Science and Technology, Volume 10, Issue 1 (Ver 1.0), Pages 18-22, April 2010.

[6] Simon T. Powers, Jun He, “A hybrid artificial immune system and Self Organizing Map for network intrusion detection”, Information Sciences: an International Journal, Volume 178, Issue 15, Pages 3024-3042, August 2008.

[7] Xiaorong Cheng, Shanshan Wen, “A real-time hybrid intrusion detection system based on Principle Component Analysis and Self Organizing Maps”, Sixth International Conference on Natural Computation (ICNC), Issue Date: 10-12 Aug. 2010, Pages 1182 – 1185, September 2010.

[8] Olusegun Folorunso, Oluwatobi O. Akande, Adewale O. Ogunde, Olufunke R. Vincent, “ID-SOMGA: A Self Organizing Migrating Genetic Algorithm-Based Solution for Intrusion Detection ”, Computer and Information Science, www.ccsenet.org/cis, Vol. 3, No. 4, November 2010.

[9] Ritu Ranjani Singh, Neetesh Gupta, Shiv Kumar, “To Reduce the False Alarm in Intrusion Detection System using Self Organizing Map”, International Journal of Soft Computing and Engineering (IJSCE), ISSN: 2231-2307, Volume-1, Issue-2, May 2011.

[10] Soewito B, Vespa L, Mahajan A, Weng N, Wang H., “Self addressable memory-based fsm (sam-fsm): a scalable intrusion detection engine”, IEEE Network Special Issue on Recent Developments in Network Intrusion Detection 2009; 23(1): 14–21.

[11] Snort, Snort Rule Database, 2007. Available at: http:// www.snort.org/pub-bin/downloads.cgi.

[12] Van Lunteren J.,”High-performance pattern-matching for intrusion detection”, INFOCOM 2006. 25th IEEE International Conference on Computer Communications, Proceedings, 2006; 1–13.

[13] Lin Tan, Timothy Sherwood, “A High Throughput String Matching Architecture for Intrusion Detection and Prevention”, In ISCA ’05: Proceedings of the 32nd Annual International Symposium on Computer Architecture, Washington DC, USA, 2005; 112–122 (IEEE Computer Society).

[14] Lin Tan, Timothy Sherwood, “Architectures for bit-split string scanning in intrusion detection”. IEEE Micro 2006; (1): 2–9.

[15] Brodie B, Cytron R, Taylor D. “A scalable architecture for high-throughput regular-expression pattern matching”, In Proceedings of 33rd International Symposium on Computer Architecture, 2006.

[16] Casas, P., Mazel, J., Owezarski, P. “Knowledge-independent traffic monitoring: Unsupervised detection of network attacks”, IEEE Network, Volume 26, Issue 1, pages 13 – 21, January 2012.

[17] Alok Tongaonkar, Sreenaath Vasudevan, and R. Sekar, “Fast packet classification for Snort by native compilation of rules ”, LISA'08 Proceedings of the 22nd conference on Large installation system administration conference, Pages 159-165.

[18] Ning Weng, Lucas Vespa, Benfano Soewito, “Deep Packet Pre-filtering and finite state encoding for adaptive intrusion detection system”, Computer Networks, Volume 55, Issue 8, Pages 1631-2022, June 2011.

[19] Gidiya Priyanka V., Ushir Kishori N, Mirza Shoeb A, Ikhankar Sagar D, Khivsara Bhavana A., “ A Proposed System for Network Intrusion Detection System Using Data Mining”, IJCA Proceedings on International Conference in Computational Intelligence (ICCIA2012), iccia - Number 8, March 2012, Published by Foundation of Computer Science, New York, USA.

[20] Bei Qi, Yun Feng Dong, “A New Model of Intrusion Detection Based on Data Warehouse and Data Mining”, Advanced Materials Research, Volumes 383 – 390, Pages 303-307, November, 2011.

[21] Kumar, Sailesh. "Survey of current network intrusion detection techniques." (2007): 1-18.

[22] A. V. Aho and M. J. Corasick, "Efficient string matching: An aid to bibliographic search," Comm. of the ACM, 18(6):333-340, 1975.

[23] B. Commentz-Walter, "A string matching algorithm fast on the average," Proc. of ICALP, pages 118-132, July 1979.

[24] S. Wu, U. Manber," A fast algorithm for multi-pattern searching," Tech. R. TR-94-17, Dept. of Comp. Science, Univ of Arizona, 1994.

[25] Snorby Data, https://www.snorby.org, Last accessed July, 2014.

[26] GitHub.com files, https://github.com/firnsy/barnyard2, Last accessed July, 2014.

[27] Matthew Tanase, The Future of IDS, 2002. http://www.securityfocus.com/infocus/1518

[28] S. Kumar et al., "Algorithms to accelerate multiple regular expressions matching for deep packet inspection," Proc. ACM SIGCOMM, 2005.

[29] Dhawal Thakker, Choosing the right intrusion detection system, 2003. http://www.expresscomputeronline.com/20030929/security16.shtml

[30] GitHub.com files, https://github.com/firnsy/barnyard2/blob/master/schemas/create_mysql, Last accessed August, 2014.

[31] KDD Cup 1999 Data, http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html, Last accessed July, 2014.

[32] Intrusion Detection Overview, http://www.pearsonitcertification.com/articles/article.aspx?p=174342, Last accessed August, 2014.

[33] Dissecting Snort, http://www.pearsonhighered.com/samplechapter/157870281X.pdf, Last accessed August, 2014.

[34 Snort files, http://manual.snort.org/node31.html, Last accessed July, 2014.

[35] Snorby Data, http://www.aldeid.com/wiki/Snorby, Last accessed July, 2014.

Authors

Md. Azizul Islam completed B.Sc. in Computer Science and Engineering and M.Sc. in Computer Science from American International University-Bangladesh.Currently working as an Infrastructure Security & Data Center Specialist in IT Division of Meghna Bank Limited, Dhaka, Bangladesh. Research interests include NIDS, WSN, Core Network Design, Planning and Implementation.

Md. Manirul Islam received his B.Sc. in Computer Engineering from University of Baguio and MSc. in IT from Saint Louis University. Currently, he is serving as an Assistant Professor under Faculty of Science and Information Technology and Director, Continuing Education Center at American International University -Bangladesh (AIUB). His research interests include Network Intrusion Detection and Wireless Sensor Networks.