Using different processors as predictors to detect a data packet logging into the Snort network with minimal delay time

Suhad Abbas Yassir¹, Nahla Flayyih Hasani² and Haidar Raad Shakir³
¹Technical Institute Shattra, Iraq.
²College of basic education/University of Sumer, Iraq.
³College of physical education and science/University of Thi-Qar, Iraq.

Abstract. The growing number of Internet threats has increased demand for better defence and information security in computer systems. Snort is a Network Intrusion Detection System (NIDS) used in network security by staying on the edge of a network and performing deep packet checking on any packet logging into the protected domain. The question of how much overhead is introduced into the network traffic by the introduction of real-time NIDS was thus investigated. A simulation methodology was used to implement experiments to evaluate the effect of Snort, measured by end-to-end delay-time introduced by the engine. These experiments proved that there is no noticeable effect on network traffic from such an introduction. IBM SPSS software version 24 was used in this study.

Keywords: NIDS, Defence In-Depth, Snort, Overhead

1. Introduction

Network Security is a large and growing area of concern for all corporations that have computers connected to the internet. As the number of companies with computers and services accessible to the Internet increases, so does the number of attacks against companies. Furthermore, there is as yet no mechanism that can promise to totally secure a network. Intrusion Detection refers to identifying individuals who are using a computer system without authorization (i.e., crackers) and those who have legitimate access to the system but are abusing their privileges (i.e., insider threat). Intrusion Detection Systems (IDSs) have thus evolved into a critical component in secure network architecture, and an IDS is now any hardware, software, or combination thereof that monitors a system or network of systems for malicious activity, as defined by Koziol [1][2].

Intrusion Detection Systems (IDSs) are classified by functionality and can be loosely grouped into three categories: Network IDS, Host IDS, and Distributed IDS. NIDS monitor traffic as it flows through a network, while HIDS reside on a particular host and monitor for intrusion attempts; DIDS is a combination of NIDSs and HIDSs as appropriate across the enterprise, with all parts reporting to a central correlation system [3, 4].

Network intrusion detection systems (NIDS) are a major security component in many network environments. These systems continuously monitor network traffic for malicious activity, raising
alerts when they detect attacks, thus enabling real-time detection of network attacks [5]. When using network intrusion detection systems, network administrators need to ensure that network traffic is unduly delayed by overhead introduced by the real-time network intrusion detection system, however. Network administrators cannot either endanger network security or add unneeded overhead to their already extremely busy networks by introducing a slow network detection system. There are two broad categories analysis performed to look for signs of intrusion. The first is misuse detection, which works by applying knowledge accumulated about specific attack types and system vulnerabilities. The intrusion-detection system contains information about these vulnerabilities and looks for attempts to exploit them; when such an attempt is detected, an alarm is triggered. The second type of analysis is anomaly detection. Anomaly detection techniques assume that an intrusion can be detected by observing deviations from normal or expected behaviour in the system or its users [6, 7].

End-to-end method of delay measurement was used by Cisco in 2007 to test for effects on the network when using IDS. These systems have the potential to introduce delay, and generally, as these networks will carry voice, video or multimedia traffic, as well as data, standards for delay limits must be established to determine when the delay value becomes unacceptable (table 1) [8].

| RANGE IN MILLISECONDS | DESCRIPTION |
|------------------------|-------------|
| 0-150 µs               | Acceptable for most user applications. |
| 150-400 µs             | Acceptable provided administrators are aware of the transmission time and the impact this has on the transmission quality of user applications. |
| Above 400 µs           | Unacceptable for general network planning purposes. However, in some exceptional cases, this limit may be exceeded. |

2. Snort

Snort was designed by Martin Roesch in 1998. It is a free, cross-platform, lightweight network intrusion detection tool [9] that can be used to monitor small TCP/IP networks, and it is capable of performing real-time traffic analysis and packet logging on IP networks [10]. It can also perform protocol analysis and content searching/matching, and it can be used to detect a variety of attacks and probes, such as buffer overflows, stealth portscans, CGI attacks, SMB probes, and OS fingerprinting attempts [11].

Snort is primarily a misuse-based NIDS that uses a combination of rules and pre-processors to analyse traffic [12]. Snort uses a flexible rules language to describe which traffic it should collect or pass, as well as a detection engine that utilises modular plugin architecture. The pre-processor code allows examination that is more extensive and manipulation of data that cannot be done via the rules alone [4].

Snort can run in three modes [13], which makes it very powerful; these are packet sniffing, packet logging, and intrusion detection system. Packet sniffing mode simply reads the packets off the network and displays them in a continuous stream on the console, while packet logger mode logs these packets to the disk. Network intrusion detection mode is the most complex and configurable, however, allowing Snort to analyse network traffic for matches against a user-defined rule set as well as to perform several actions based upon what it thus identifies.

Snort is logically divided into multiple components. These components (see figure 1) work together to detect particular attack types and to generate output in a required format from the detection system.
Figure 1: Snort Architecture.

The most important feature is the use of Snort in IDS mode. Snort is a packet sniffer that is designed to take packets and process them through pre-processors. Each packet observed on the network is first passed through a set of pre-processors, which may extract information and/or modify the packet and then those packets are checked against a series of rules within the detection engine. Then detection plug-ins match the packet against signature conditions, and if a match is found, the alert system is triggered to allow any issues to be handled by whatever plug-ins is chosen to handle such alerts [2,4].

2.1 Snort Pre-processors

A pre-processor is a code that is compiled into the Snort engine upon build in order to normalise traffic and/or examine the traffic for attacks beyond the level possible in the normal rules. This verges on being an overly simplistic explanation for the functions of these complex pieces of Snort, but it emphasises their contribution to the overall intrusion detection system (IDS) [4, 14].

Snort allows selection of which pre-processors should be enabled through the Snort configuration file “snort.conf” [15]. Snort has many pre-processors available. The Snort project team has certified some, while others remain in testing and still more are in development. These pre-processors are what make Snort such a powerful and effective intrusion detection system. The pre-processors that this paper examines are the Frag3, Stream5, Http-Inspect, Ftp-Telnet, and sfPortscan pre-processors.

2.1.1 Frag3. The frag3 pre-processor reassembles packets, acting as a target-based IP defragmentation module for Snort [16]. Target-based analysis is a relatively new concept in network-based intrusion detection, and the idea of a target-based system is to model the actual targets on the network instead of only modelling the protocols and looking for attacks within them.

2.1.2 Stream5. The Stream5 pre-processor is a target-based TCP reassembly module for Snort [16]. It is intended to replace both the Stream4 and flow pre-processors, and it can track sessions for both TCP and UDP. Many attacks are spread across several packets and are undetectable to non-session-reassembling rule-matching IDS; this is thus the purpose of stream reassembly [4, 14, 17].

2.1.3 Http-Inspect. Http has become one of the most widely and diversely used protocols on the Internet. Over time, researchers have found that Web servers will often take a number of different expressions of the same URL as being equivalent [16]. For example, an IIS Web server will see these two URLs as identical:

http://www.example.com/foo/bar/iis.html  http://www.example.com/foo\bar\iis.html
Unfortunately, a pattern matcher such as Snort will only match the pattern foo/bar against the first of these. An attacker can use this “flexibility” in Web servers to attempt to hide probes and attacks from the NIDS. Thus, http_inspect is stateless, normalising HTTP strings on a packet-by-packet basis; it will thus only process HTTP strings that have been reassembled by a pre-processor, once Stream4, though this has been replaced by Stream5 [4].

2.1.4 sfPortscan. This module is designed to detect the first phase in a network attack: Reconnaissance. In the Reconnaissance phase, an attacker determines what types of network protocols or services a host supports. This is the traditionally where a portscan takes place. This phase assumes that the attacking host has no prior knowledge of the protocols or services supported by the target, otherwise this phase would not be necessary [16].

2.1.5 Ftp-Telnet Pre-processor. Ftp_telnet is composed of two parts: the FTP pre-processor and the telnet pre-processor. Overall, ftp_telnet can be stateful or stateless; it receives this data from the stream4 pre-processor, which when replaced by stream5 must be combined with the ftp-telnet pre-processor [4].

When a telnet data buffer is available, ftp_telnet normalises the buffer with respect to telnet commands and option negotiation, eliminating telnet command sequences per RFC 854. When FTP command channel buffers (on port 21) are used, ftp_telnet interprets the data, identifying FTP commands and parameters, as well as determining the appropriate FTP response codes and messages. This enforces the correctness of the parameters, determines when an FTP command connection is encrypted, and, furthermore, determines when an FTP data channel is opened [16]. Ftp_telnet is thus extremely versatile, having the capability through its dynamic pre-processor to configure every parameter, creating a very powerful emulation engine.

2.2 Advantages and Disadvantages of Snort

Snort is a very flexible application. Due to its modular design and the ability to add specialised software components Snort can be used as a powerful tool in any in-depth defence and security implementation. The design allows anyone capable of programming to build and implement their own pre-processor modules, customising Snort’s operations to their specific environment. Customisation can also be accomplished through specialised configurations of the existing pre-processor modules, as well as by changing alert output operations [4].

Snort has a large following and, according to the Snort website snort.org, Snort has effectively become the standard in intrusion detection systems. There are many commercial systems available, but many organisations use Snort because it offers an effective intrusion detection system while remaining free of charge. Snort is a signature-based detection system, and its large user base means that new signatures are constantly being added. This large user and support base has thus generated a highly effective and efficient detection engine [16].

Snort does have some limitations in anomaly detection, however. The system was not designed for this type of operation, although some pre-processor modules attempt to add this functionality [18]. Currently, these modules are not considered to be effective in detection. There is also concern about how efficient the detection engine actually is in terms of processing performance. The base engine is considered quite efficient, but there is speculation as to how efficient the system becomes when used with these pre-processor modules. The added functionality is thus good, but may come at a price in terms of processing [16].

3. Experiment framework

An experiment was designed to evaluate Snort performance in terms of its effect on its environment. A simulation methodology was used to implement experiments in an attempt to answer the following question "How much overhead, measured by elapsed time or delay time in network
traffic, is introduced by the implementation of a real-time intrusion detection system?”. The null hypothesis for this question is that the delay added by the intrusion detection system is not noticeable.

3.1 Assumptions

While testing the effects of the Snort intrusion detection system, the scope of the test identified by certain indications of performance. The test and performance measurements thus had to be controlled and protected from the effects of other processing components. To ensure this, the following assumptions were made.

i. Assumption 1

Snort is an intrusion detection system designed to detect network intrusions through both pattern matching and detection of anomalous network behaviour. For this study, it was thus assumed that Snort is capable of performing both functions efficiently and effectively.

ii. Assumption 2

Snort can be installed and used within a few minutes of installation. However, there are many customisable components in the Snort system that were not considered for this research. This study uses only the default configuration, assuming that this is sufficiently optimised for basic testing to determine delay or elapsed time from end-to-end traffic introduced by the intrusion detection system.

iii. Assumption 3

That the sample traffic used for testing the intrusion detection system is representative of normal network traffic on the live network.

3.2 Scope and Limitations

The tests were implemented in an isolated local area network (LAN) using a Windows operating system. An isolated LAN was use as many of the tests required direct control over the amount of computing activity in the environment. This research was thus limited by the following boundaries:

- This study was limited to select pre-processor modules used by the Snort intrusion detection system. There are several such modules available for Snort; some have been tested and verified for enterprise usage, while others are still being developed and tested. This study examined the Frag3, Stream5, Http-Inspect, Ftp-Telnet, and sfPortscan pre-processor modules available in Snort.
- Several different methods for alert output are available in Snort. This study limited the alert options to the standard default, which processes alerts to a basic log file. The system alert output was not reviewed for this study, as this information determines detection accuracy, not speed.

3.3 Simulation Model

To get a better idea of how the detection engine operates in a live environment, traffic was tested in a simulated version. Such tests can be quite disruptive to the network, so although traffic needs to be obtaining from a live network, a test environment must be set up to ensure better control under this type of scenario.

To prepare a test bed, three computers were required. The machines used in this study are identical, except that one machine had a second network interface card (NIC) installed configured as a bridge to allow network traffic to pass through. The machines all contained Dual-Core 1.83GHz processors, 512 RAM, and Realtek RTL8168/8111 PCI-E Gigabit network interface cards. The additional NIC was an Intel Pro/ 100+ management adapter. Each machine was loaded with Windows XP, and the machines were named IDSSource, IDS, and IDSDest, dependent on
their purpose in the simulation network. All three machines also had Wireshark and WinPcap software installed. IDSSource additionally had the Colasoft Packet Player application, while IDS had Snort 2.8.1 software installed. IDS had two network interface cards installed, and this was used for the network intrusion detection system engine. The experiment procedures were designed to test Snort’s monitoring of a network computer, but the best environment to use for these tests was deemed to be an isolated area network as the tests require direct control over the amount of computing activity in the environment.

The sample test traffic was captured using the Wireshark packet sniffer application from a live network and saved in file named Test.pcap. The Colasoft Packet Player was then used to read this file and to send the packets on a specified network interface with the exact timing for when the traffic was captured recorded. This ensured that the data used was as close to live traffic as possible. The data then left the IDSSource machine and was sent to IDSDest, although it had to cross IDS before reaching its destination. The IDS computer thus set the Snort sensor on the bridge connection where the packets were to be processed and then forwarded on to IDSDest. Once the packet was received at IDSDest, it was read by the Wireshark application.

3.4 Experiment Design

Regression was used to predict the smallest delay time for detecting a data packet logging on to the Snort network. This work thus performed extraction of the morality amount, with the results sent through ANOVA to determine the t and f variances.

As numerous possible combinations of pre-processors may be used for this purpose, this study used a full-factorial experimental design to consider five pre-processor modules; throughout, each module could be turned either on or off. The pre-processor modules were Frag3, Stream5, Http-Inspect, Ftp-Telnet, and, sfPortscan, though the last three modules require a Stream5 pre-processor for activation. This creates 17 possible combinations, which thus required 18 test passes. To help ensure that unknown factors did not affect the results, each test was performed with ten replications to allow determination of how each pre-processor module combination affects the total end-to-end delay in network traffic; by averaging multiple passes that may encounter various noise effects, the effect of any such noise is reduced, providing results that are more reliable.

4. Analysis experimental results

A feasibility study was done to predict the smallest delay time for detecting a data packet logging onto the Snort network.

Statistical analysis was performed on data from the Snort processors that indicated that using different processors simultaneously can detect a data packet logging onto the Snort network with the least delay time. This statistical analysis was applied to the results obtained with regard to the amount of delay time required to detect a data packet logging into the Snort network for each processor individually and with other Snort processors. The results of this analysis showed that the least delay was realised when Stream5 and http-inspect processors are used simultaneously.

In tables 2, 3, 4, 5, and 6, the time delays depending on the Snort processors are laid out to determine the preferred options for increased performance of the Snort network, with the aim being to detect a data packet in near real time.
Table 2. ANOVA\(^a\) and Coefficients\(^a\) Dependent on variable Steam5

| Model       | Sum of Squares | df | Mean Square | F     | Sig. |
|-------------|----------------|----|-------------|-------|------|
| Regression  | .000           | 1  | .000        | 5.716 | .033\(^b\) |
| Residual    | .000           | 13 | .000        |       |      |
| Total       | .001           | 14 |             |       |      |

\(a\). Dependent Variable: Steam5  
\(b\). Predictors: (Constant), Http-Inspect

Table 3. ANOVA\(^a\) and Coefficients\(^a\) Dependent on variable Frag3

| Model       | Sum of Squares | df | Mean Square | F     | Sig. |
|-------------|----------------|----|-------------|-------|------|
| Regression  | .001           | 1  | .001        | 4.928 | .045\(^b\) |
| Residual    | .004           | 13 | .000        |       |      |
| Total       | .005           | 14 |             |       |      |

\(a\). Dependent Variable: Frag3  
\(b\). Predictors: (Constant), Stream5, Http-Inspect, Ftp-Telnet, Sfportscan

Table 3. ANOVA\(^a\) and Coefficients\(^a\) Dependent on variable Frag3

| Model                        | Unstandardized Coefficients | Standardized Coefficients | t     | Sig. |
|------------------------------|-----------------------------|---------------------------|-------|------|
| (Constant)                   | .065                        | .004                      | 16.743| .000 |
| Http Inspect                | -.190                       | .080                      | -.553 | -2.391 | .033 |

\(a\). Dependent Variable: Frag3  
\(b\). \(Y= 0.12795\)
Table 4. ANOVA\(^a\) and Coefficients\(^a\) Dependent on variable Http-Inspect

| Model     | Sum of Squares | df | Mean Square | F     | Sig.  |
|-----------|----------------|----|-------------|-------|-------|
| Regression| .001           | 1  | .001        | 6.382 | .025\(^b\) |
| Residual  | .003           | 13 | .000        |       |       |
| Total     | .005           | 14 |             |       |       |

a. Dependent Variable: Http-Inspect  
b. Predictors: (Constant), Stream5, Http-Inspect

| Model             | Unstandardized Coefficients | Standardized Coefficients | t    | Sig.  |
|-------------------|-----------------------------|---------------------------|------|-------|
| (Constant)        | .103                        | .014                      | 7.260| .000  |
| Stream5 Frag3     | -.688                       | .272                      | -2.526| .025  |

a. Dependent Variable: Http-Inspect  
b. \(Y = 1.273\)

Table 5. ANOVA\(^a\) and Coefficients\(^a\) Dependent on variable Sfportscan

| Model     | Sum of Squares | df | Mean Square | F     | Sig.  |
|-----------|----------------|----|-------------|-------|-------|
| Regression| .001           | 1  | .001        | 5.124 | .041\(^b\) |
| Residual  | .003           | 13 | .000        |       |       |
| Total     | .004           | 14 |             |       |       |

a. Dependent Variable: Sfportscan  
b. Predictors: (Constant), Stream5, Http-Inspect, Ftp-Telnet

| Model             | Unstandardized Coefficients | Standardized Coefficients | t    | Sig.  |
|-------------------|-----------------------------|---------------------------|------|-------|
| (Constant)        | .056                        | .005                      | 11.292| .000  |
| Stream5 Frag3 Http-Inspect | -.078                   | .034                      | -2.264| .041  |

a. Dependent Variable: Sfportscan  
b. \(Y = 0.178\)
### Table 6. ANOVA\(^a\) and Coefficients\(^a\) Dependent on variable Ftp-Telnet

| Model       | Sum of Squares | df | Mean Square | F     | Sig.  |
|-------------|----------------|----|-------------|-------|-------|
| Regression  | .001           | 1  | .001        | 4.304 | .058\(^b\) |
| Residual    | .004           | 13 | .000        |       |       |
| Total       | .005           | 14 |             |       |       |

\(a\). Dependent Variable: Ftp-Telnet  
\(b\). Predictors: (Constant), Steam5

| Model     | Unstandardized Coefficients | Standardized Coefficients | t    | Sig.  |
|-----------|------------------------------|---------------------------|------|-------|
| (Constant)| .073                         |                            | 6.797| .000  |
| Frag3Strems5 | -.430                   | -.499                     | -2.075| .058  |

\(a\). Dependent Variable: Ftp-Telnet  
\(b\). \( Y = 0.787 \)

### 5. Conclusions

In this study, regression was used to predict the delay time for the detection of a data packet logging onto the Snort network. This work then extracted the morality values for ANOVA, generating the t and f variance for the tested processes.

Using Stream5 and http-inspect processors simultaneously in the Snort network realised a minimal delay time of 0.058986858 \(\mu s\) for the detection of data packets logging onto the network.
6. References

[1] Nancy Agarwal and Syed Zeeshan Hussain 2018 Security and Communication Networks A Closer Look at Intrusion Detection System for Web Applications Volume 2018 Article ID 9601357 27 pages.

[2] Kozio J. 2003 Intrusion Detection with Snort: Sams Publishing.

[3] Baker A. R. Caswell B. and Poor M. 2004 Snort 2.1 Intrusion Detection 2nd ed. USA: Syngress Publishing.

[4] Baker A. R. and Esler J. 2007 Snort IDS and IPS Toolkit Burlington: Syngress Publishing.

[5] Attig M. and Lockwood J. 2005 Proc. of IEEE COMPUTER SOCIETY SIFT: Snort Intrusion Filter for TCP.

[6] Wu Y.-S., Foo B., Mei Y., and Bagchi S. 2003, Proc. of 19th Annual Computer Security Applications Conf. (ACSAC 2003), IEEE, Collaborative Intrusion Detection System (CIDS): A Framework for Accurate and Efficient IDS.

[7] Debar H., Dacier M., and Wespi A., 1999, Towards a Taxonomy of Intrusion Detection Systems, Elsevier.

[8] Cisco 2007 Understanding Delay in Packet Voice Networks, Retrieved from http://www.cisco.com/warp/public/788/voip/delay-details.html.

[9] Roesch M. 1999 Snort - Lightweight Intrusion Detection for Networks Retrieved from http://www.snort.org/docs/lisapaper.txt.

[10] Caruso L. C., Guuindani G., Schmitt h., Neycalazans, and Moraes F. 2007 Proc. of 18th IEEE/IFIP Int. Workshop on Rapid System Prototyping (RSP’07) IEEE SPP-NIDS - A Sea of Processors Platform for Network Intrusion Detection Systems.

[11] Hutchings B. L., Franklin R. and Carver D. 2002 Proc. of 10th Annual IEEE Symp. on Field-Programmable Custom Computing Machines (FCCM’02) Assisting Network Intrusion Detection with Reconfigurable Hardware.

[12] Sourdis I . Dimopoulos V., Pnevmatikatos D., and Vassiliadis S. 2006 ACM pp. 183–192 Packet Pre-filtering for Network Intrusion Detection.

[13] Guerrero J. H. and Cardenas R. G. 2005 An example of communication between security tools: Iptables - Snort ACM pp. 34–43.

[14] Beale J. and Foster J. C. 2003, USA: Syngress Publishing Snort 2.0 Intrusion Detection.

[15] Rehman R. U. 2003 1st ed. New Jersey Printice Hall PTR Intrusion Detection Systems with Snort.

[16] Snort 2008 Snort Users Manual 2.8.2.

[17] Novak J. and Sturges S. 2007 Target-Based TCP Stream Reassembly Sourcefire Incorporated.

[18] Lippmann R., Haines J. W., Fried D. J., Korba J., and Das K. 2000 The 1999 DARPA Off-Line Intrusion Detection Evaluation Lincoln Laboratory MIT.