# Overhead Evaluation in Real-Time Network Intrusion Detection System Using Snort

Suhad Abbas Yasir
Technical Institute / shattra

***Abstract***

A growing number of Internet threats have increased the need of applying a defense in depth concepts to protect the information contained on computer systems worldwide. Snort is a lightweight Network Intrusion Detection System (NIDS) that widely used in network security. However, to ensure that such an implementation is likely to be successful, the system must be tested to provide decision makers with assurance to reduce risks.

Typically, NIDS residing on the edge of a network performs deep packet inspection on every packet that enters to the protected domain. Real-Time NIDS obviously place some additional overhead into the network traffic path. How much overhead introduced into the network traffic by introducing of real-time NIDS. A simulation methodology had been used to implements some experiments designed to evaluate Snort effect, measured by end-to-end delay-time introduced by the engine. These experiments proved that there is no noticeable effect introduced to the network traffic.

*NIDS, Defence in-Depth, Snort, Overhead*

## I. INTRODUCTION

Network Security is a large and growing area of concern for corporations that have computers connected to the internet. At the same time as the number of companies with computers and services accessible to the Internet increases, so does the number of attacks against companies. Furthermore, up to now there is no mechanism that can promise to totally secure a network. [1]

Intrusion Detection has been defined by [1] as “the problem of 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., the ‘insider threat’)”.

Intrusion Detection Systems (IDSs) have evolved into a critical component in secure network architecture. An IDS is any hardware, software, or combination of thereof that monitor a system or network of systems for malicious activity as defined by koziol [2].

Intrusion Detection Systems (IDSs) are classified by there functionality, loosely grouped into three categories: Network IDS, Host IDS and Distributed IDS. NIDS monitor traffic as it flows throw a network; HIDS are reside on a particular host and monitors for intrusion attempts; and DIDS is a combination of NIDSs, HIDSs, or both across the enterprise and all reporting to 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 and also enable real-time detection of network attack [5]. With the use of network intrusion detection systems, a network administrators need to ensure that network traffic is not being unduly delayed by overhead introduced from the real-time network intrusion detection system. Network administrators do not want to endanger network security or add unneeded overhead to the already extremely busy networks by introducing a network detection system.

There are two broad categories analysis performed to look for signs of intrusion. The first is misuse detection. Misuse detection works by apply the knowledge accumulated about specific attacks 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 performed is anomaly detection. Anomaly detection techniques assume that an intrusion can be detected by observing a deviation from normal or expected behavior of the system or the users [6, 7].

End-to-end method of delay measurement was used by Cisco in 2007 to test effect on the network when using IDS. These systems have the potential to introduce delay. Generally the networks will similarly carry voice, video or multimedia traffic, as well as data, needs to be some standards for delay limits established. Such standards have been established to assist in determining when the delay value becomes unacceptable (refer to table 1). These standards suggest that overall per packet delay should remain below 150 milliseconds to ensure acceptable network performance [8].

**Table 1:** Delay Specifications.

| RANGE IN MILLISECONDS | DESCRIPTION |
|---|---|

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

## 2- SNORT

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

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

Snort can run in three modes [13] that make it very powerful: 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. Packet logger mode logs the packets to the disk. Network intrusion detection mode is the most complex and configurable; allowing Snort to analyze network traffic for matches against a user-defined rule set and to perform several actions based upon what it sees.

Snort is logically divided into multiple components. These components (refer to figure 1) work together to detect particular attacks and to generate output in a required format from the detection system.

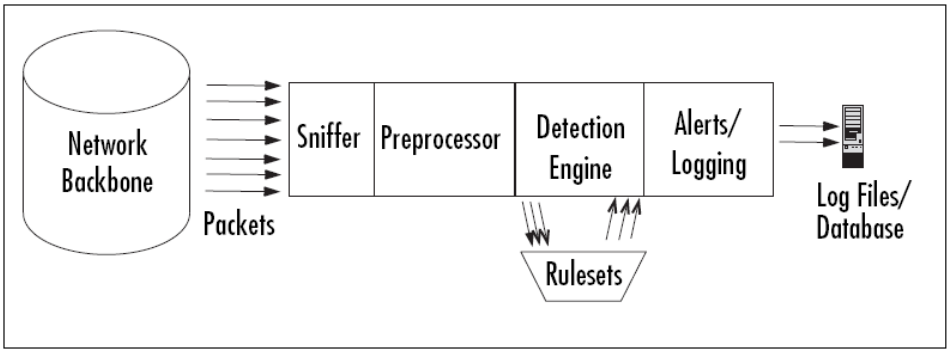

Figure 1: Snort Architecture.

The most important feature is using Snort as IDS mode. Snort is a packet sniffer. However, it is designed to take packets and process them through the preprocessors. Each packet observed on the network is first passed through a set of preprocessors, which may extract information and/or modify the packet and then check those packets against a series of rules (through the detection engine). Then detection plug-ins matches the packet against signature conditions. If a match was found, sent through the alert system, it can be handled by whatever plug-ins has been chosen to handle alerting [2,4].

### 2.1-1 Snort Preprocessors

A preprocessor is a code that is compiled into the Snort engine upon build in order to normalize traffic and/or examine the traffic for attacks in a fashion beyond what can be done in normal rules. Although that might seem like an overly simplistic explanation for what these complex pieces of Snort do, it's important to realize their contribution to the overall whole of the intrusion detection system (IDS) [4, 14].

Snort allows us to select which preprocessors should be enabled. From this standpoint, this is done through the Snort configuration file "snort.conf" [15]. Snort has many preprocessors available. The Snort project team has certified some, while others are in testing and more yet are still in development. These preprocessors are what make Snort such a powerful and effective intrusion detection system. The preprocessors that we are primarily concerned with this paper are the Frag3, Stream5, Http-Inspect, Ftp-Telnet, and sfPortscan.

#### *2-1`-2 Frag3*

The frag3 preprocessor is applying for reassembling packets as a target-based IP defragmentation module for Snort [16]. Target-based analysis is a relatively new concept in network-based intrusion detection. The idea of a target-based system is to model the actual targets on the network instead of only modeling the protocols and looking for attacks within them.

### 2.1.2 Stream5

The Stream5 preprocessor is a target-based TCP reassembly module for Snort [16]. It is intended to replace both the Stream4 and flow preprocessors, and it is capable of tracking sessions for both TCP and UDP. Many attacks are spread across several packets and are undetectable to a non session-reassembling rule-matching IDS, that's the whole reason for 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 equivalent [16]. For example, an IIS Web server will see these two URLs as being 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 two. An attacker can use this "flexibility" in the Web server to attempt to hide his probes and attacks from the NIDS. *http_inspect* is stateless; it normalizes HTTP strings on a packet-by-packet basis and will only process HTTP strings that have been reassembled by the *Stream4* which replaced by *Strean5* preprocessor [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 traditional place where a portscan takes place. This phase assumes the attacking host has no prior knowledge of what protocols or services are supported by the target, otherwise this phase would not be necessary[16].

### 2.1.5 Ftp-Telnet Preprocessor

*Ftp_telnet* is composed of two parts: the *FTP* preprocessor and the *telnet* preprocessor. *ftp_telnet* can be stateful or stateless; it receives this data from the *stream4* preprocessor which replaced by *stream5*, thus we need to combine it with *ftp-telnet* preprocessor [4].

When a telnet data buffer available, *ftp_telnet* will normalize 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* will interpret the data, identifying FTP commands and parameters, as well as appropriate FTP response codes and messages. It will enforce the correctness of the parameters, determine when an FTP command connection is encrypted, and furthermore determine when an FTP data channel is opened [16]. *ftp_telnet* is extremely versatile, having the

capability through the *dynamic* preprocessor to be able to configure every last parameter, which makes for a very powerful emulation engine.

## 2.2 Advantages and Disadvantages of Snort

Snort is a very flexible application. Due to the modular design and ability to add or break in specialized software components Snort can be a powerful tool in a defense/security in-depth implementation. This design allows anyone capable of programming to build and implement their own preprocessor modules to customize Snort's operation to their specific environment. Customization can also be accomplished through specialized configurations of the existing pre-processor modules, as well as alert output operations [4].

Snort also has a large following and according to the Snort website snort.org, Snort is the effectively standard in intrusion detection systems. There are many commercialized systems available, but many organizations use Snort because it is an effective intrusion detection system, and free cost. Snort is a signature based detection system and with the large user base new signatures are constantly being added. This large user and support base has led to what is described as a highly effective and efficient detection engine [16].

Snort does have some limited shortfalls when it comes to anomaly detection. The system was not designed for this type of operation, but some pre-processor modules attempt to add this functionality [18]. Currently these modules are not considered 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 the pre-processor modules. The added functionality is good, but what price do you have to pay for that functionality [16].

# 3- THE EXPERIMENT FRAMEWORK

An experiment was designed to evaluate the snort performance from its effect on the environment viewpoint. A simulation methodology had been used to implement these experiments. This research will attempt to answer the following question "How much overhead, measured by elapsed time or delay time in network traffic, will be introduced by the implementation of a real-time intrusion detection system?". The null hypothesis for this question is the delay added by the intrusion detection system will not be noticeable.

## 3.1 Assumptions

While testing effect of the Snort intrusion detection system, the scope of the test is specific to certain indications of performance. The test and

performance measurements must then be controlled and should not be effected by other processing components of the software. To ensure this the following assumptions will be made.

*i. Assumption 1*

Snort is an intrusion detection system that is designed to detect network intrusions through both pattern matching and detection of anomalous network behavior. For this study, it will be 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 customizable components designed in the Snort system that will not be considered for this research. This study will use the default configuration assuming that it is sufficiently optimized for basic testing to determine the amount of delay or elapsed time from end-to-end traffic introduced by the intrusion detection system.

*iii. Assumption 3*

It will also be assumed that the sample traffic used for testing the intrusion detection system will be representative of normal network traffic on the live network.

## 3.2 Scope and Limitations

The tests will be implements in isolated local area network (LAN) using windows operating system. Isolated LAN will use because many of the tests required direct control over the amount of computing activity in the environment. This research will be limiting by the following boundaries:

- This study will be limited to the study of 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. We will limit this study to the Frag3, Stream5, Http-Inspect, Ftp-Telnet, and sfPortscan preprocessor modules available in Snort.
- Several different methods for alert output are available in Snort. This study will limit the alert options to the standard default, which processes alerts to a basic log file. The system alert output will not be reviewed for this study, as this information would be use to determine detection accuracy.

## 3.3 Design the Simulation Model

To get a better idea of how the detection engine operates in a live environment this traffic should be tested in it. This can be quite disruptive

to the network, so traffic needs to be obtaining from a live network that can be use in a test environment. Better control of the testing environment can be maintained using this type of scenario.

To prepare a test bed, setting three computer computers will be required. The machines being used in this study will be identical, one machine will have a second network interface card (NIC) installed that need to be configure as a bridge to allow network traffic to pass through them. The machines contain Dual-Core 1.83GHz, 512 RAM, and Realtek RTL8168/8111 PCI-E Gigabit network interface cards. The additional NIC was Intel Pro/ 100+ management adapter. Each machine loaded with windows XP, these machines will be called IDSSource, IDS, and IDSDest depend on the machine's purpose on the simulation network. All the three machines will also require having the Wirshark and WinPcap softwares installed. IDSSource will have the Colasoft Packet Player application, while IDS will have the Snort 2.8.1 software installed. Machine IDS will have two network interface cards installed, as it will be used for the network intrusion detection system engine. The experiment procedures are designed for testing Snort that monitors a network computer. The best environment to use for these tests is isolated area networks because of the tests require direct control over the amount of computing activity in the environment.

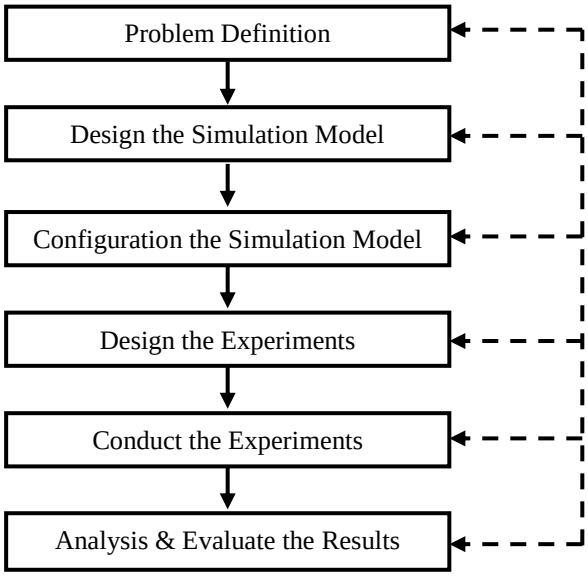


**Figure 2**: Simulation Test-Bed Model

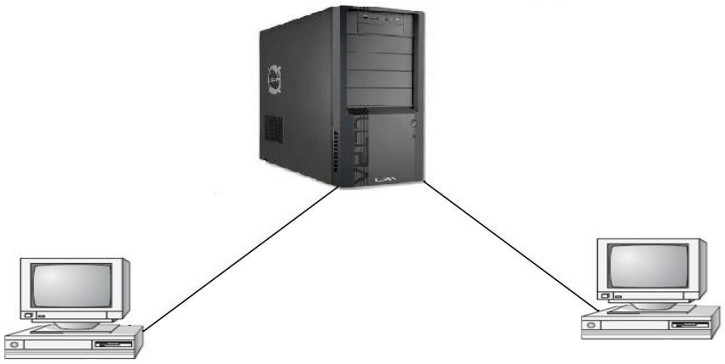

**Figure 3**: Intrusion Detection Simulated Network

The sample test traffic captured using the Wireshark packet sniffer application from a live network and will be saved in file named Test.pcap. The Colasoft Packet Player will be used to read the file Test.pcap and then send the packets on a specified network interface using the exact timing recorded when the traffic was captured. This assists in ensuring that the data being used is as close to live traffic as possible. The data will then leave the IDSSource machine and it will be sent to IDSDest. However, it must be cross IDS before reaching its destination. IDS computer set the Snort sensor's on the bridge connection where the packets will be processed and then it will be forwarded on to IDSDest. Once the packet is received at IDSDest it will be read by the Wireshark application.

### 3.4 Design the Experiments

Due to the numerous possible combinations of preprocessors that may be used, this study will use a full-factorial experimental design. Under consideration are five preprocessor modules and each module can be turned either on or off. The pre-processor modules are Frag3, Stream5, Http-Inspect, Ftp-Telnet and, sfPortscan, the last three modules need to turn on with Stream5 preprocessor. This indicates that there are 17 possible combinations, which require 18 test passes. To help ensure that unknown factors do not affect the results, each test will be performed with ten replications. The replications will allow a determination of how each preprocessor module combination affects the total end-to-end delay of the network traffic by averaging multiple passes that may encounter noise effects, thus reducing the effect of the noise and providing results that are more reliable. using a simulation scripts to generate data conforming to the statistical distribution in a real environment; third, collected from a real environment and replicated in the experimental environment as described by wan & yang in (2001).

## 4- ANALYSIS EXPERIMENTAL RESULTS

Once the data has been collected, an analysis will be made of the obtained results. To start, a statistical analysis will be performed on the results obtained to determine the amount of overhead introduced into the end-to-end delay by the intrusion detection system.

The observations showing the average value per treatment run and the delay introduced per packet listed in Table 2. As well, examine the practical significance of the end-to-end delay time introduced by the Real-Time Intrusion Detection System will be made. This will be

determined by looking at the per-packet delay (refer to Table 2). The delays calculated will be compared to the information in Table 1 to see if any treatment factor combination will generate a per packet delay greater than 150 µs.
Reviewing these data shows that the delay time introduced by using SNORT Real-Time NIDS per packet ranged from 58.988883 µs to 58.986858 µs. This overhead is well below the 150 µs mark for acceptable delay in end-to-end traffic.
Table 2: End-to-End Delay Time Introduced by IDS Engine per Packet.

Total Packets Number (N) = 8254
Delay (per Packet) = Average Observe Time / N

| **Preprocessors Options** | **Average Observe Time** | **Delay/packet** |
|---|---|---|
| Frag3*Stream5 | 486.894243 | 0.058988883 |
| Stream5 | 486.891141 | 0.058988508 |
| Frag3 | 486.889177 | 0.05898827 |
| Frag3*Stream5*Http-Inspect | 486.884414 | 0.058987693 |
| Frag3*Stream5*Ftp-Telnet | 486.8833637 | 0.058987565 |
| Stream5*Http-Inspect*Ftp-Telnet | 486.883207 | 0.058987546 |
| Frag3*Stream5*Http-Inspect*Ftp-Telnet | 486.8830503 | 0.058987527 |
| Stream5*sfPortscan | 486.8828936 | 0.058987508 |
| Frag3*Stream5*sfPortscan | 486.8827369 | 0.058987489 |
| Stream5*Http-Inspect*sfPortscan | 486.8825801 | 0.05898747 |
| Stream5*Ftp-Telnet*sfPortscan | 486.8824234 | 0.058987451 |
| Frag3*Stream5*Http-Inspect *sfPortscan | 486.8822667 | 0.058987432 |
| Frag3*Stream5*Ftp-Telnet *sfPortscan | 486.88211 | 0.058987413 |
| Stream5*http-Inspect*Ftp-Telnet* sfPortscan | 486.8819533 | 0.058987394 |
| Frag3*Stream5*Http-Inspect*Ftp-Telnet * sfPortscan | 486.8817966 | 0.058987375 |
| Stream5*Ftp-Telnet | 486.87884 | 0.058987017 |
| Stream5*Http-Inspect | 486.877529 | 0.058986858 |

## 5- CONCLUTIONS

Through the methodology followed, try to answer the question "how much end-to-end delay is introduced by the implementation of Snort as a real-time network intrusion detection system". Using the Snort in a real-time network intrusion detection system and considering the many various combinations of options and configurations the amount of end-to-end delay per packet will range from 58.988883 µs to 58.986858 µs. This is a very small in relative terms and would be very difficult to notice in a

normal operating environment. It is clear that, delay per packet is considerably less than the 150 µs, which means that there is no practical significance added to the end-to-end delay. The conclusion is to fail to reject the null hypothesis.

Overall, there is a good indication that the Snort is capable of carrying the load of a production network with minimal statistical impact and virtually no practical impact. This is an unexpected result as much larger delays were thought to be introduced by the real-time intrusion detection system associated with a variance between treatment factors. The Snort pre-processors obviously require processing cycles to complete their work, which in theory should increase the end-to-end delay of network traffic. However, the preprocessors save processor cycles by reducing the amount of traffic that must pass through the pattern matching engine, offsetting the increased processing cycles, creating a very operationally efficient security tool.

## REFERANCES


[1] Mukherjee B., Heberlein L. T., and Levitt K. N., "Network Intrusion Detection", *IEEE Network*, vol. 8, pp. 26-41, 1994.

[2] koziol j., *Intrusion Detection with Snort*: Sams Publishing, 2003.

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

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

[5] Attig M. and Lockwood J., "SIFT:Snort Intrusion Filter for TCP," Proc. of IEEE COMPUTER SOCIETY2005.

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

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

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

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

[10] Caruso L. C., Guuindani G., Schmitt H., neycalazans, and Moraes F., "SPP-NIDS - A Sea of Processors Platform for Network Intrusion Detection Systems," Proc. of 18th IEEE/IFIP International Workshop on Rapid System Prototyping(RSP07), IEEE, 2007,

[11] Hutchings B. L., Franklin R., and Carver D., "Assisting Network Intrusion Detection with Reconfigurable Hardware", Proc. of 10th

Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'02), 2002.
[12] Sourdis I. , Dimopoulos V., Pnevmatikatos D. , and Vassiliadis S. , "Packet Pre-filtering for Network Intrusion Detection", *ACM*, pp. 183 -192, 2006.
[13] Guerrero J. H. and Cardenas R. G., "An example of communication between security tools: Iptables - Snort," *ACM*, pp. 34 - 43, 2005.
[14] Beale J. and Foster J. C. , *Snort 2.0 Intrusion Detection*. USA: Syngress Publishing, 2003.
[15] Rehman R. U., *Intrusion Detection Systems with Snort*, 1st ed. New Jersey Printice Hall PTR, 2003.
[16] Snort, "Snort Users Manual 2.8.2", 2008.
[17] Novak J. and Sturges S., "Target-Based TCP Stream Reassembly," Sourcefire Incorporated, 2007.
[18] Lippmann R.,. Haines J. W, Fried D. J., Korba J., and Das K., "The 1999 DARPA Off-Line Intrusion Detection Evaluation," *Lincoln Laboratory MIT*, 2000.

## الملخص

إن الاهتمام والوصول إلى النتائج العملية التي يولي لها المختصون في مجال الشبكات هوكيفية التقليل من الاختراقات الموجودة على الشبكة حيث يعد ( snort) برنامج لاكتشاف الاختراق في الشبكة وهو من البرامج الواسعة في أمنية الشبكات ، ولغرض التأكد من نجاح استخدامه يجب اختباره بشكل الذي يضمن سلامة الشبكة من الاختراقات من خلال هذا الاختبار الذي يتيح لكل العاملين في هذا المجال العمل من خلال تزويدهم بالنتائج الايجابية .وغالبا تستخدم نظم اكتشاف واختراق الشبكة في أطراف الشبكة لكي تقوم بالفحص الدقيق لحزم البيانات الداخلة إلى المنطقة المحمية ، وان استخدام نظم الاكتشاف الاختراق في الشبكة في الوقت الحقيقي يضيف حمل أضافي إلى الشبكة ، لذلك تم اعتماد طريقة المحاكاة لتنفيذ بعض التجارب لتقييم عمل تأثير برنامج snort مقاسا بوقت التأخير ( end-to- end ) على أطراف الشبكة . وان هذه التجارب المتمثلة بمعالجات رئيسية أثبتت بعدم تأثير محسوس على سير البيانات في الشبكة نتيجة استخدام هذا البرنامج .