Smart Logic - Preventing Packet Loss in High Speed Network Intrusion Detection Systems

Ahsan Subhan, Monis Akhlaq, Faeiz Alserhani, Irfan U. Awan, John Mellor, Andrea J. Cullen, and *Pravin Mirchandani

Informatics Research Institute, University of Bradford,
Bradford BD7 1DP, United Kingdom and
*Syphan Technologies (www.Syphan.com)
{s.a.subhan,m.akhlaq2,f.m.f.alserhani,i.u.awan,j.e.mellor,
a.j.cullen}@Bradford.ac.uk, pmirchandani@Syphan.com

Abstract. Network Intrusion Detection Systems (NIDS) have gained substantial importance in today's network security infrastructure. The performance of these devices in modern day traffic conditions is however found limited. It has been observed that the systems could hardly stand effective for the bandwidth of few hundred mega bits per second. Packet drop has been considered as the major bottleneck in the performance. We have identified a strong performance limitation of an open source Intrusion Detection System (IDS), Snort in [1, 2]. Snort was found dependent on host machine configuration. The response of Snort under heavy traffic conditions has opened a debate on its implementation and usage. We have developed the Smart Logic component to reduce the impact of packet drop in NIDS when subjected to heavy traffic volume. The proposed architecture utilizes packet capturing techniques applied at various processing stages shared between NIDS and packet handling applications. The designed architecture regains the lost traffic by a comparison between the analysed packets and the input stream using Smart Logic. The recaptured packets are then re-evaluated by a serialized IDS mechanism thus reducing impact of packet loss incurred in the routine implementation. The designed architecture has been implemented and tested on a scalable and sophisticated test bench replicating modern day network traffic. Our effort has shown noticeable improvement in the performance of Snort and has significantly improved its detection capacity.

Keywords: Network intrusion detection systems, network performance, packet drop, Snort, serialization.

1 Introduction

Intrusion detection in the realm of information security is the technique of identifying any hostile attempt into the network. The concept has gained acceptance at all levels on account of increasing threats for users/ networks. It has also been complemented by various mechanisms to fulfill security requirements. Broadly Intrusion Detection systems (IDS) are categorized into two types. Signature based – these are designed to

D. Weerasinghe (Ed.): ISDF 2009, LNICST 41, pp. 57–65, 2010.
© Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering 2010

detect attacks on basis of well defined threats, these are accounted into system policies to prevent any future intrusion. Anomaly based - These base their logic on pre defined behavioral patterns where any violations from the defined routine generate an alert [4]. Both the techniques have inherent advantages and disadvantages. The anomaly based systems can detect undiscovered attacks; however they are suffered by large number of false positive. On the other hand signature based mechanisms cannot detect new attacks thus need continuous updates. We have based our research on signature based detection techniques on basis of low false positive, better performance and a greater acceptance in the domain of network security.

In modern day network traffic, the performance of these systems has been found debatable. It has been observed that the optimal potential of these systems has not been explored. A wide gap still exists between the capability of these systems and recent developments in hardware/peripheral devices. For example the systems still fails to maximize performance by using multi-core architecture. It also suffers from heavy packet loss once used in Giga-bit Ethernet environment.

Numerous efforts have been made to address the issues related to packet loss in high speed networks. Quite few techniques focus on securing a balance between the detection capacity of system and input traffic. Few substantially competent techniques make use of load balancing concepts. These involve the use of traffic splitting mechanism where input traffic is distributed across the set of detection engines for evaluation and filtering to block the traffic destined to unpublished ports.

Krugel et al in [5] explored a parallel architecture to increase the system capacity by splitting traffic into manageable sized slices; these are then fed to the set of detection sensor. Each sensor has been made intelligent to respond to set of particular threats and has no intercommunication. Unfortunately, this technique is strongly dependent on few parameters. Firstly, the sliced traffic must be fed to the detection engine capable of responding to the particular threat which in some cases is violated and may result in sensor overloading; thus resulting in packet loss. Secondly, it may fail to partition complex stateful signatures since correct partitioning of the traffic is known only at runtime. Thus any approach that uses a static partitioning algorithm needs to over-approximate the event space associated with the signature resulting in improper partitioning [6].

The parallel architecture for stateful detection described in [6] also based its logics on splitting the traffic and distributing it to detection sensors in a round robin fashion. It differs from [5] on grounds that sensors are able to communicate each other via control node on a high-speed, low-latency, dedicated control plane and all have the similar rule matching directory. The feedback allows the sensors to synchronize their scanning process. The stateful detection is ensured by replicating the matcher state which is same for all sensors at any time. Unfortunately, the concept too suffers from performance limitations. Storing already scanned packets in the buffer for a possibly pending evaluation in another sensor creates overhead. Response to slow attacks would also be an issue where every sensor would be maintaining the state and would keep on occupying buffer space. The use of a centralized node for synchronization creates overhead and consumes processing power.

The concept of Early Filtering (EF) applies to the filtering of incoming traffic before evaluation to handle heavy input as described in [7] is also found limited in performance. The technique uses filtering as a part of sensor functionality. The input is first evaluated through EF rule-set. If matched, the packets are discarded; otherwise

analyzed. The EF rule-set belongs to the main rule directory and has comparatively a less complex analysis, for example, inspection of the header only. This could be easily evaded by the attempts, concealing payload in initial packets and bypassing the evaluation stage. Using locality buffers in the second stage creates performance bottleneck once subjected to heavy input traffic when packets awaiting session completions cross the buffer limit.

Our effort adopts a different approach. We base our mechanism on the serialization concept where input traffic is being analyzed by normal sensors and later the dropped traffic is recovered by our Smart Logic. The technique is distinguished in a way that limitation observed in the traffic splitting approaches and drawbacks in early filtering are avoided to improve the performance. The Smart Logic takes an approach using capture, identify and evaluate cycle with an RDBMS [8] acting as a common data repository. Periodic data cleansing has also been applied to ensure optimal performance of the database. We have implemented our logic on Snort [9], an open source NIDS analyzing real world traffic.

The paper is organized into sections. Section 2 describes the architecture and algorithm of Smart Logic. Section 3 & 4 gives the test-bench and results. Finally in the conclusion we analyze our contribution.

2 Smart Logic

2.1 Architecture

The architecture is based on Serialization Concept [1]. This incorporates two instances of IDS evaluation. The instances are separated by Smart Logic which is implemented to recover dropped packets by active comparison as shown in Figure 1. The first instance deals with the traffic directly fed from Ethernet interface and carries out evaluation. The output of first instance is fed to the data-store where it gets compared with the duplicated input using Smart Logic. The comparison of these two identifies the lost packets. These lost packets are then fed to the second stage serialized evaluation process.

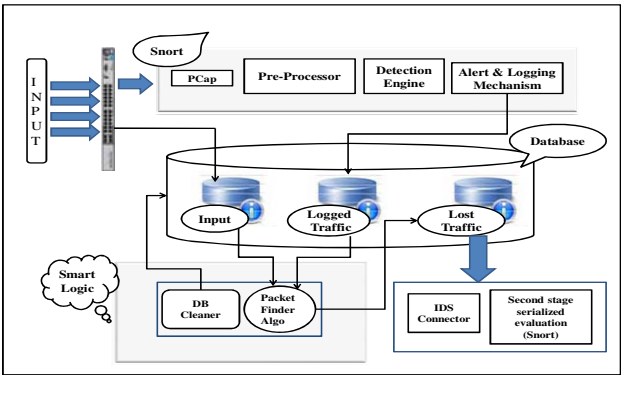

Fig. 1. System Architecture

The system is composed of two components Snort and Smart Logic. Snort, an open source NIDS has been selected because of its popularity and status as a de facto IDS standard. Snort relies on the packet capturing libraries. The libraries act as packet capturing interfaces to the underlying operating system [10].

Snort is an open signature based IDS; it uses rules to detect hostile attempts into a network. The rules are the set of requirements created to generate an alert and have a particular syntax. On successful detection of a hostile intrusion, the detection engine sends an alert to a log file/storage resource. The Smart Logic operates by collection of network traffic through multiple channels (Snort alert logging mechanism and Ethernet span port) and executes comparison logic to identify the lost packets. The lost packets are then re-evaluated by Snort based internal detection engine as shown in Figure 1. Shared data-store has been employed to act as a bridge between Snort and Smart Logic. We have also utilized Sharp PCap [11] wrapper around libpcap [10] for managed languages like C# [12], VB.NET [13] and Boo [14] on various frameworks such as Microsoft .NET [15] and MONO [16]. Sharp PCap has been installed on server and shall ensure the Smart Logic to respond to both open source and proprietary operating systems environments.

2.2 Operation

Packet capturing and handling is a vital requirement for a NIDS. A good packet capturing response by the system towards variant traffic reduces the probability of a system becoming compromised. Factors that affect the packet capturing performance of an NIDS in a Gigabit Ethernet environment include host configuration parameters (hardware and software) and application-specific parameters (NIDS). We have developed comparison logic - Smart Logic to address the issue of packet loss in high speed gigabit ethernet environment.

The designed algorithm – Smart Logic recovers the lost traffic by active comparison of input traffic with analyzed packets in the first stage. The operation unfolds in two steps, initialization and execution.

2.2.1 Initialization

Sharp PCap requires to be present on the server, Smart Logic extracts the current operating system time stamp. Ethernet interface is checked to be functional and status of Snort is verified to be active. The algorithm initializes the logic by setting parameters such as packet receipt threshold (example 1000 milliseconds). This delay ensures that the component terminates its operation if no traffic is detected at the network interface for 1000 milliseconds.

2.2.2 Execution

On successful initialization, the logic executes analysis cycles. The procedure iterates as long as the Ethernet Interface receives packets or the Snort process is active. The concept of Smart Logic is based on re-evaluation of lost traffic. This has been ensured by using a shared data-store for segregating input traffic and analyzed traffic. The difference of the input and analyzed traffic identify the dropped packets. These packets are then fed to a Snort based internal detection engine for re-evaluation as shown in Figure 2. The execution cycle terminates automatically once Snort process made inactive or once the variable packet receiving time out expires.

A periodic operational cycle is implemented to perform data purging on the data-store connected to the system. The data-store plays a vital role in the comparative analysis of network traffic. Smart Logic reads the current system time stamp and after every selected time period (example 15 minutes) it filters the lost traffic and purges the data-store.

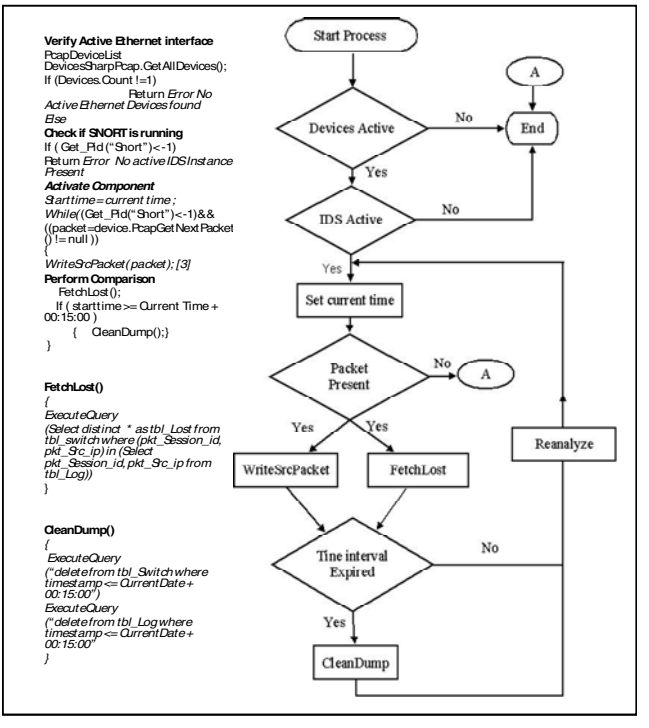

Fig. 2. Smart Logic Process Flow

Our algorithm executes a loop that iterates until one or more exit conditions met (termination of Snort process or packet receive threshold reached). In each iteration, the algorithm stores the input traffic and checks for the expiry of execution interval (15 minutes). On expiry of the interval, traffic filtering and purging are executed within the iteration.

The purging facility executes in parallel with the lost packet fetch routine (in a producer-consumer paradigm). The multiple delete operations remove the consumed input and analyzed packets from their respective tables *(Input traffic - tbl_Switch and analysed traffic – tbl_Log)* at fixed time intervals.

The traffic filtering component of Smart Logic classifies lost packets on the basis of session ID, source and destination IP addresses. The lost packets are identified by comparing incoming traffic captured by Sharp PCap to the packets analyzed by Snort available in the data-store. Difference of these shall give the dropped packets. Mathematically the Smart Logic has implemented as follows:

Let $D_T = T$ (D_T is the Traffic logged in the database from the network interface

Let x be the input traffic fed to first stage Snort instance S1 where $x = T$

$$\text{First stage evaluation} \longrightarrow S_1(x) \xrightarrow{\text{yields}} T_a \quad \text{where } T_a \in T \tag{1}$$

$$\text{Smart Logic} \longrightarrow T_d = D_T / T_a \quad \text{where } T_d \text{ is the dropped traffic} \tag{2}$$

$$\text{Second stage Snort instance} \longrightarrow S_2(T_d) \xrightarrow{\text{yields}} T_f \tag{3}$$

$$\text{Total evaluated traffic (T)} \longrightarrow T = T_a + T_f \tag{4}$$

3 Test Bench

The network is composed of six machines using ProCurve Series 2900 Switch [17] as shown in Figure 3. The test bench comprises high performance PCs running open source tools to generate background traffic [18, 19] and monitor network performance. The network is quite similar to that of used in [1 , 2] with the exception of Smart Logic component.

Snort was analyzed by using Smart Logic under different traffic conditions, operating system platforms and system configurations. We have selected two important parameters (System CPU Usage and Packet handling) to evaluate the proposed mechanism.

4 Results

We have evaluated the performance of Snort under different scenarios and traffic condition in [1, 2]. It was found the Snort barely stands effective in Giga bit Ethernet environments. The main factor for the limited system performance relates to the large number of packet drop. In order to identify the efficacy of our concept we have compared our results with ones' obtained in [1, 2]. Performance of the system in context of CPU usage and packet handling capability is described in following paragraphs.

Due to paucity of space we have included the results from Linux 2.6 platform only, we have also identified in the previous efforts that performance of Snort is better in Linux in comparison to others.

4.1 CPU Usage

Application performance largely depends upon CPU usage, higher the usage compromised would be the performance. In [1, 2] we have identified that a single instance of Snort consumes 20 – 40 % of the CPU once subjected to input traffic volume ranging from 200 Mbps – 2.0 Gbps as shown in Figure 4.

Implementation of Smart Logic and executing second stage serialized Snort increased the CPU usage upto 70 % for traffic volume of upto 1.2 Gbps as shown in Figure 4. Above 1.2 Gbps, CPU usage increased to 100% (2.0 Gbps input traffic) as

shown in Figure 4. The increase in CPU usage affects the performance of system and cause packet drop.

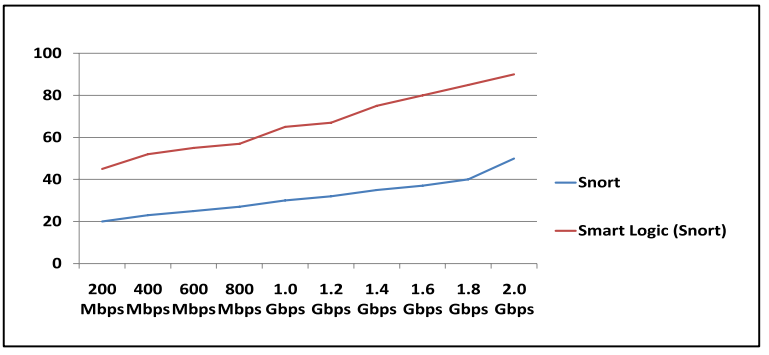

Fig. 4. Comparison – CPU Usage

4.2 Packet Handling

We have achieved a 100% processing response with no packet drop using Smart Logic. This is a significant improvement from a standard Snort which can barely handle input traffic reaching 500 Mbps as shown in Figure 5. Snort start dropping packets once bandwidth increased from 200 Mbps and at 2.0 Gbps it dropped more than 70% of the input traffic.

Our serialized implementation of evaluation process runs two instances of Snort. This has improved the overall evaluation process and successfully able to analyse traffic upto 1.2 Gbps as shown in Figure 5. The system start dropping packets above 1.2 Gbps, the basic cause of packet loss incurred in the evaluation is the bottleneck caused by a low disk data transfer rate.

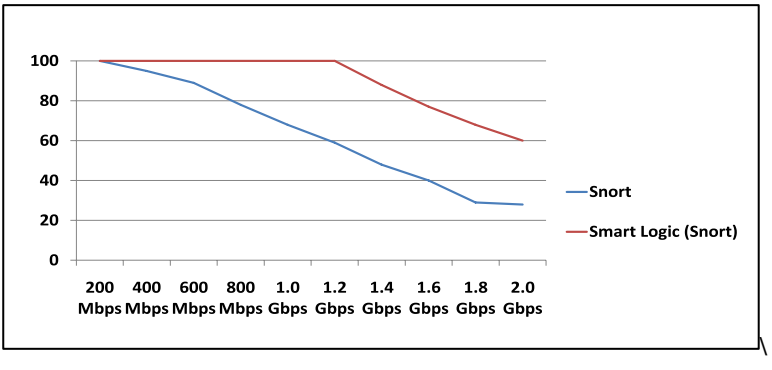

Fig. 5. Comparison – Packets Analyzed

5 Conclusion

We have proposed a serialized IDS evaluation concept to utilize system resource and mitigate the packet loss. Smart Logic designed to recover lost packets has shown considerable improvement in the packet handling capability of the system. The introduced concept manages two instances of Snort and still operates within system capability. The technique also fulfills the requirements of real time traffic analysis.

The multi-threaded approach of design has improved the network utilization, this has also influenced on the processing capability of the system. The IDS implemented in the proposed algorithm can analyze greater volume traffic and reduce processing delays.

The data purging concept used in the Smart Logic would execute multiple delete operation to clean the data-store at specified interval. This ensures managed depth of data-tables thus allowing optimized data access.

In order to handle traffic above 1.2 Gbps, we suggest running of second Snort instance on a different machine. We have performed few tests on this configuration and have found system responsive for input traffic volume up to 5.0 Gbps.

We have managed to eliminate the performance limitations relative to the use of load balancing concepts in traffic splitting mechanisms. This has been achieved by monitored duplication and multistage analysis of network traffic. The use of a shared data repository ensures minimal packet loss and detection efficiency while simultaneously reducing performance overhead.

Acknowledgment

We would like to thank Dr. D. R. W. Holton and Dr. R. J. Fretwell of University of Bradford for their guidance and support during the course of this research. We would also appreciate Syphan Technologies for providing us the opportunity to work on the SINBIN Test Lab and their assistance throughout.

References

1. Alserhani, F., Akhlaq, M., Awan, I., Cullen, A., Mellor, J., Mirchandani, P.: Evaluating Intrusion Detection Systems in High Speed Networks. In: Fifth International Conference of Information Assurance and Security (IAS 2009), August 18-20. IEEE Computer Society, Xian (in press, 2009)
2. Alserhani, F., Akhlaq, M., et al.: Snort Performance Evaluation. In: Proceedings of Twenty Fifth UK Performance Engineering Workshop (UKPEW 2009), Leeds, UK, July 6-7 (2009)
3. Kazienko, P., Dorosz, P.: Intrusion detection systems (IDS) Part 2 - Classification; methods; techniques (2004)
4. Tessel, J.D., Young, S., Linder, F.: The Hackers Handbook. Auerbach Publications, New York (2004)
5. Krugel, C., Valeur, F., vigna, G., Kemmerer, R.: Stateful Intrusion Detection for High Speed Networks. In: Proceedings of IEEE Symposium on Security and Privacy, Oakland, CA, May 2002, pp. 285–293 (2002)

6. Fischini, L., Thapial, A.V., Cavallaro, L., Kruegel, C., Vigna, G.: A Parallel Architecture for Stateful, High-Speed Intrusion Detection. In: Proceedings of fourth International Conference on Information system security, Hyderabad, India, pp. 203–220 (2008)
7. Xinidis, K., Charitakis, I., Antonatos, S., Anagnostakis, K.G., Markatos, E.P.: An Active Splitter Architecture for Intrusion Detection and Prevention. IEEE Trans. Dependable Sec. Computer 3(1), 31–44 (2006)
8. RDBMS, http://www.databasedir.com/what-is-rdbms
9. Snort, http://www.Snort.org
10. Baker, A.R., Esler, J.: Snort IDS and IPS Toolkit, Syngress, Canada (2007)
11. Sharp PCap, http://www.chrishowie.com/pcap-sharp
12. C Sharp, http://en.wikipedia.org/wiki/C_Sharp
13. VB.net, http://vb.net
14. Boo, http://boo.codehaus.org
15. Microsoft.Net, http://www.microsoft.com/NET
16. MONO, http://mono-project.com/Main_Page
17. VMware Server, http://www.vmware.com/products/server
18. LAN Traffic V 2, http://www.topshareware.com/lan-traffic-v2/downloads/1.html
19. D-ITG V 2.6, http://www.grid.unina.it/Traffic/index.php