DEPLOYMENT OF INTRUSION PREVENTION SYSTEM ON MULTI-CORE PROCESSOR BASED SECURITY HARDWARE

Swetha K V[1] and Ravi Dara[2]

[1]Department of Computer Science & Engineering, CMR Institute of Technology, Bangalore, India
[2]Nevis Networks(I) Pvt.Ltd., Pune, India

ABSTRACT

After tightening up network perimeter for dealing with external threats, organizations have woken up to the threats from inside Local Area Networks (LAN) over the past several years. It is thus important to design and implement LAN security strategies in order to secure assets on LAN by filtering traffic and thereby protecting them from malicious access and insider attacks. Banking Financial Services and Insurance (BFSI) industry is one such segment that faces increased risks and security challenges. The typical architecture of this segment includes several thousands of users connecting from various branches over Wide Area Network (WAN) links crossing national and international boundaries with varying network speed to access data center resources. The objective of this work is to deploy LAN security solution to protect the data center located at headquarters from the end user machines. A LAN security solution should ideally provide Network Access Control (NAC) along with cleaning (securing) the traffic going through it. Traffic cleaning itself includes various features like firewall, intrusion detection/prevention, traffic anomaly detection, validation of asset ownership etc. LANenforcer (LE) is a device deployed in front of the data center such that the traffic from end-user machines necessarily passes through it so that it can enforce security. The goal of this system is to enhance the security features of a LANenforcer security system with Intrusion Prevention System (IPS) to enable it to detect and prevent malicious network activities. IPS is plugged into the packet path based on the configuration in such a way that the entire traffic passes through the IPS on LE.

KEYWORDS

LAN security, LANenforcer, IPS, Security hardware, Multi-core processor

1. INTRODUCTION

LAN security solutions are important for the complete protection of enterprise networks and the users on the network. The security solutions include firewalls, anti-virus programs, Intrusion Detection/Prevention Systems (IDPS), traffic anomaly detection mechanisms that identify attacks as it occurs. Firewalls, anti-virus, and IDS helps prevent confidential data from getting out and also prevent intruders getting in. A firewall filters traffic from the Internet into the organization as well as traffic from within the organization to the outside. IDS can evaluate traffic that passes through these open ports but cannot stop it. Network -based IPS is generally systems that sit in line, and block suspicious traffic after detecting an attack. IPS protects networks from unauthorized network connections, malicious network activities and intruders. In IPS mode, device is not working with copy of the packets, but instead, it is working with the original packets. IPS has become an essential next-level of defence for environments that want transparency to users while protecting data and network resources. IPS may be implemented in hardware or in software on a PC. The various IPS software's use different detection methods, signature detection, anomaly detection, and some proprietary methods to prevent the occurrences of attacks. IPS is an evolution of IDS technology. Its proactive capabilities will help to keep networks safer from more sophisticated attacks. IPS focuses on what an attack does — its behavior, which does not change. IPS use a set of rules to represent the type of behavior: acceptable or harmful. The real time traffic is then compared to the set of rules and the action is taken, whether to be permitted or blocked.

The challenges associated with financial organizations possess increased economical risk at every step of transactions. The challenges include data access control and security, availability of network connectivity and so on. Security threats and attacks can end up in disrupting the entire network including ATMs. To have a control on access by unknown machines and monitor for detection of malicious behavior within the traffic and thereby block the detected traffic, it is necessary to adopt LAN security strategies.

1.1. Problem Statement

Most of the segments in the current world face many risks and security challenges. Banking Financial Services and Insurance (BFSI) industry is one among them. The typical infrastructure of BFSI industry includes the large number of users with connections from various branches over WAN links. The Number of branches varies depending upon the organization and network speed varies depending upon the infrastructure. Critical servers are located at the Headquarters. LEs are deployed in front of data centers such that the traffic from end-user machines necessarily passes through it so that it can enforce security. Critical servers need to be protected from end-users and machines. LE with LAN security solutions such as user authentication, validation of asset ownership, MAC address verification, NAC, firewall, traffic anomaly keeps networks safer from network- based attacks. These are deployed in n:m redundancy model, where n is the number of active LEs and m is the number of standby LEs to ensure complete availability of the network even if connectivity to active LE fails. In this work, IPS feature is proposed to enhance the security features of a LE security system which can prevent from suspicious threats. IPS is plugged into the packet path based on configuration in such a way that the entire traffic passes through the IPS on LE. LS is a monitoring and configuring tool for LE where GUI resides. LE and LS do secure communication. IPS configuration is integrated into LS for admin to control LE.

2. THE MULTI-CORE PROCESSOR

The multi-core processor used is Cavium Network's Octeon processor. It provides high performance, high bandwidth and, low power consumption. It contains hardware acceleration for specific applications like encryptions and pattern matching. The processor can be used for control plane as well as data-plane networking applications. The Octeon processors are used in a wide variety of OEM equipment. Some examples include routers, switches, unified threat management (UTM) appliances, content-aware switches, application-aware gateways, triple-play broadband gateways, WLAN access and aggregation devices, 3G, WiMAX and LTE base station and core network equipment, storage networking equipment, storage systems, servers, and intelligent network adapters. The Octeon family of multi-core processors supports up to 32 MIPS cores. Multiple hardware acceleration units are integrated into the Octeon processor. These hardware acceleration units offload the cores, reducing software overhead and complexity. The processor consists of control plane and data plane.

3. Selection Of An Open Source Intrusion Prevention System

Intrusion Prevention Systems such as Snort, Suricata and Metaflows were studied and compared to choose a good quality and economically feasible solution [5]. Snort is a free and open source network intrusion prevention system (NIPS) and network intrusion detection system (NIDS) created by Martin Roesch in 1998. Suricata is a high performance Network IDS, IPS, and Network Security Monitoring engine. Metaflows is a product that can be installed on low- cost hardware and transmit the network data to the cloud system for analysis. It includes Bot Hunter, Snort, Flow, Net Flow like network traffic monitoring plug-ins; log management tools. Meta Flows sensors process 800 Mbps of sustained network throughput when using an eight-core Intel i7 CPU that costs around $1,000. On the server side, a threat prediction algorithm is used to prioritize events. The table below summarizes the comparison between Snort, Suricata and Metaflows.

Table 1. Comparison between snort, suricata [1],[2] and metaflows[4]

Snort	**Suricata**	**Metaflows**
Single-threaded	Multi-threaded	Multi-threaded
IDS , NIPS	NIDS , NIPS	NIDS , NIPS
Protocol analysis, content matching, TCP streaming, Signature Detection	Automated protocol detection, IP reputation, File carving, logging, built-in hardware acceleration, TCP streaming, HTTP parsing, Signature detection	Capture network events and transmit event activities to cloud system; Uses Snort underneath Behavioral malware detection, Antivirus, File carving, Signature detection, Vulnerability scanning. Saas product that can be installed on low cost hardware
Uses a dual license strategy	Free to use	Costs $99 per month for a Metaflows appliance

Based on the study, Suricata is chosen as the IPS for deployment. Suricata is a high performance Network IDS, IPS and Network Security Monitoring engine Open Source and owned by a community run non-profit foundation, the Open Information Security Foundation (OISF).

3.1. Packet Pipeline

Suricata has multiple run modes, each of which initializes the threads, queues, and plumbing necessary for operation. These modes are usually tied to the choice of the capture device and whether the mode is IDS or IPS. Example of capture devices: pcap, pcap file, nfqueue, ipfw, or a proprietary capture device. Only one run mode is chosen at startup. Each thread in the packet pipeline is an instance of a module[15],[16].

These threads[19] are initialized by the runmode defined in 'runmodes.c'. The run mode also initializes the queues and packet handlers used for moving packets between modules and queues. A thread is marked runnable after the all the steps from the run mode initialization are complete.

Real time traffic from NIC or network packets stored on PCAP file is passed as input to Suricata. Then the traffic is decoded, which converts the input in to a Suricata support data structure where it is passed to a link type decoder. Then the streams are reassembled prior to being distributed between the signature-detection modules. The detection module takes care of multiple complex tasks: loading all signatures, initializing detection plugins, creating detection groups for packet routing, and finally running packets through all applicable rules. The user can configure the number of threads, number of CPUs required in the configuration file.

Suricata is compatible with most operating systems (e.g. Linux, Mac, FreeBSD, UNIX and Windows). The industry considers Suricata a strong competitor to Snort and thus they are often compared with each other.

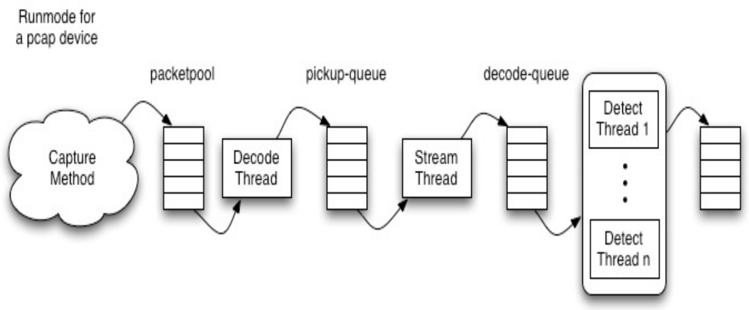


Fig 1. Packet Pipeline

3.2. Suricata.yaml file

Suricata uses the Yaml [10] format for configuration. The Suricata.yaml file is included in the source code. At the top of the YAML-file you will find % YAML 1.1. Suricata reads the file and identifies the file as YAML.

3.3. Suricata rules

Signatures play a very important role in Suricata[14]. Mostly used existing rule sets are Emerging Threats, Emerging Threats Pro and source fire's VRT[18]. A rule/signature consists of the following: The action, header and rule-options. Example of a signature:

alert udp any any -> any 53 (msg:"ET DOS DNS BIND 9 Dynamic Update DoS attempt"; byte_test:1,&,40,2; byte_test:1,>,0,5; byte_test:1,>,0,1; content:"|00 00 06|"; offset:8; content:"|c0 0c 00 ff|"; distance:2; reference:cve,2009-0696; reference:url,doc.emergingthreats.net/2009701;classtype:attempteddos; sid:2009701; rev:2;)
Description:

Col 1: action-type (alert/drop/reject/pass)
Col 2: protocol (tcp (for tcp-traffic), udp, icmp and ip. ip stands for 'all' or 'any', http, ftp, tls (this includes ssl), smb and dns)
Col 3: source ip
Col 4: source port
Col 5: direction operator (source -> destination, source <> destination (both directions))
Col 6: destination ip
Col 7: destination port
Col 8 till end: signature

4. TESTING AND OPTIMIZATION

This chapter focuses on the experiments done on LE with Suricata enabled and optimizations are done on Suricata configuration based on the results of experiments. Suricata is configured as a bridging IPS where interfaces eth0 and eth1 are bridged[17].

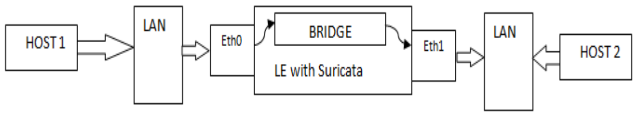


Fig.2. Experimental setup with bridge configuration

For basic testing purposes, a machine host1 is connected to LAN to which eth0 interface of LE is connected. Eth1 is connected to LAN to which another machine host2 is connected. Packets sent from host1 to host2 enter LE via eth0 interface and are forwarded to bridge, then Suricata and finally pass through eth1.

4.1. Testing using Tcpreplay and Tomahawk

Tcpreplay[11] is a tool used to replay the traffic previously captured back onto the network and through other devices such as switches, routers, firewalls, NIDS and IPSs. Tomahawk[20] is also a tool for testing the performance and in-line blocking capabilities of IPS devices. They split traffic between two interfaces and emulate client and server sides of the connection. The tool divides packet trace into two parts as those given by the client and those given by server. The First time it sees an IP address, it is classified as client if the address is found in the ip source field of packet and classified as server if the address is found in the IP destination field of the packet. Testing with these tools did not give substantial results because of the packet misbehaviour.

These tools could not do anything with the packets if it detects a source IP in the packet which is already classified as server. A DARPA Intrusion set of 400 MB and some other downloaded pcap files were replayed. But it produced warning that many packets had outgoing interface conflicts.

4.2. Testing Using Ixia

Ixia traffic generator provides a complete platform of testing the network setup to ensure sustained and reliable performance. Ixia test ports can be independently configured to define traffic, filtering, and capture capabilities. Experiments were carried out using IxExplorer and IxLoad by sending packets through DUT with Suricata enabled. Four ports of Ixia Chasis were connected to the DUT, two ports being configured as clients and two ports as servers. Using IxLoad, two client networks were configured each with 200 clients and two server networks were configured each with one server. HTTP and FTP packets were transferred using IxLoad setup. The graphs obtained as experimental results are shown below.

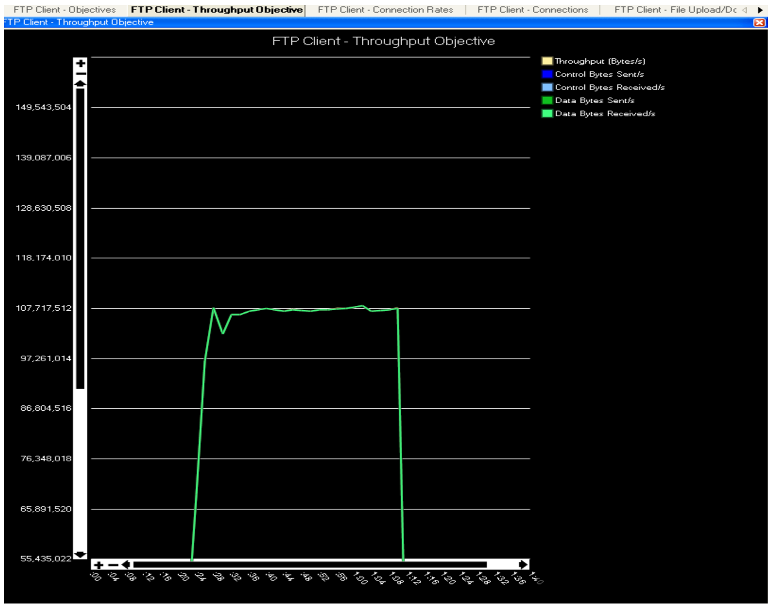


Fig.3. Graph showing throughput in MB obtained for FTP traffic through LE without Suricata

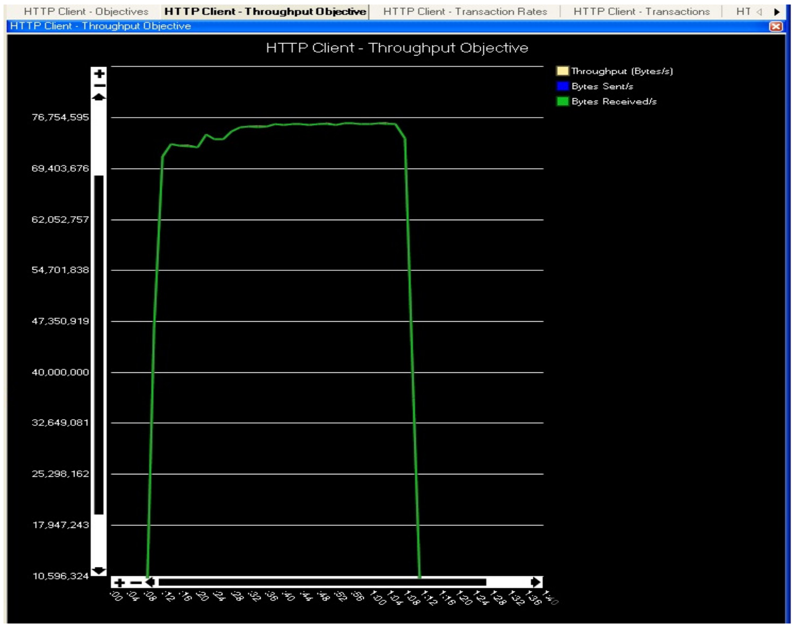


Fig.4. Graph showing throughput in MB obtained for HTTP traffic through LE without Suricata

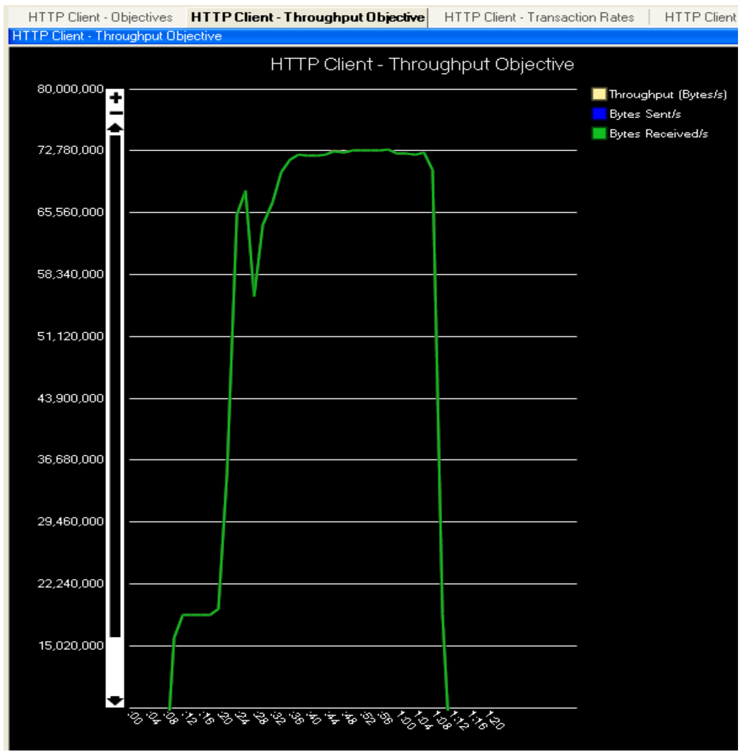


Fig.5. Graph showing throughput in MB obtained for HTTP traffic through LE with Suricata

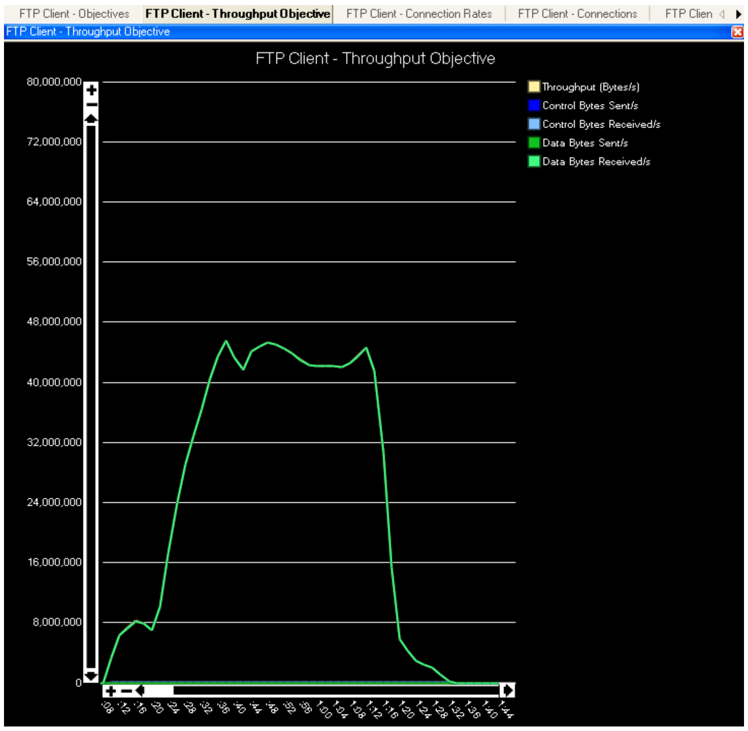
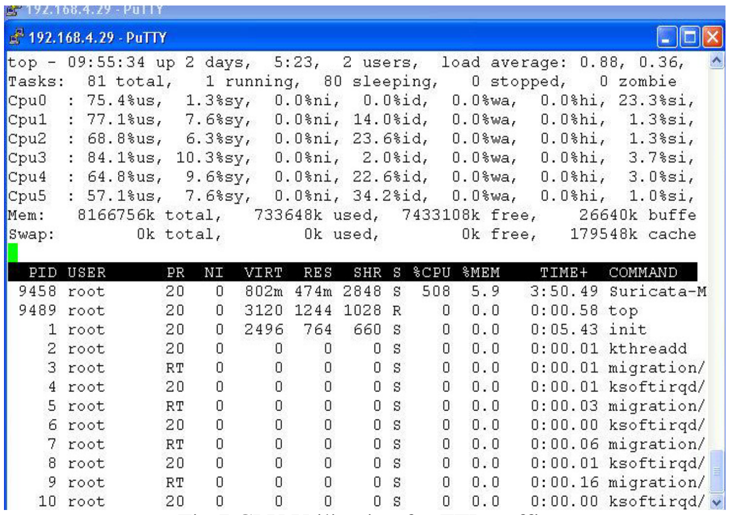


Fig.6. Graph showing throughput in MB obtained for FTP traffic through LE with Suricata

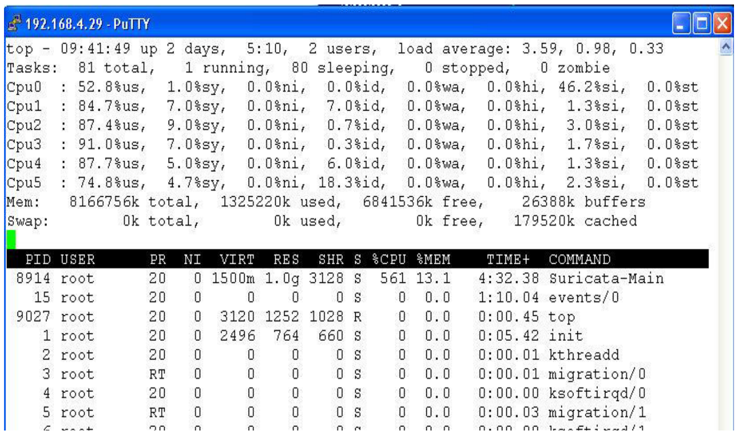


192.168.4.29 - PuTTY

```
top - 09:55:34 up 2 days,  5:23,  2 users,  load average: 0.88, 0.36,
Tasks:  81 total,   1 running,  80 sleeping,   0 stopped,   0 zombie
Cpu0  : 75.4%us,  1.3%sy,  0.0%ni,  0.0%id,  0.0%wa,  0.0%hi, 23.3%si,
Cpu1  : 77.1%us,  7.6%sy,  0.0%ni, 14.0%id,  0.0%wa,  0.0%hi,  1.3%si,
Cpu2  : 68.8%us,  6.3%sy,  0.0%ni, 23.6%id,  0.0%wa,  0.0%hi,  1.3%si,
Cpu3  : 84.1%us, 10.3%sy,  0.0%ni,  2.0%id,  0.0%wa,  0.0%hi,  3.7%si,
Cpu4  : 64.8%us,  9.6%sy,  0.0%ni, 22.6%id,  0.0%wa,  0.0%hi,  3.0%si,
Cpu5  : 57.1%us,  7.6%sy,  0.0%ni, 34.2%id,  0.0%wa,  0.0%hi,  1.0%si,
Mem:   8166756k total,   733648k used,  7433108k free,    26640k buffe
Swap:        0k total,        0k used,        0k free,   179548k cache

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 9458 root      20   0  802m 474m 2848 S  508  5.9   3:50.49 Suricata-M
 9489 root      20   0  3120 1244 1028 R    0  0.0   0:00.58 top
    1 root      20   0  2496  764  660 S    0  0.0   0:05.43 init
    2 root      20   0     0    0    0 S    0  0.0   0:00.01 kthreadd
    3 root      RT   0     0    0    0 S    0  0.0   0:00.01 migration/
    4 root      20   0     0    0    0 S    0  0.0   0:00.01 ksoftirqd/
    5 root      RT   0     0    0    0 S    0  0.0   0:00.03 migration/
    6 root      20   0     0    0    0 S    0  0.0   0:00.00 ksoftirqd/
    7 root      RT   0     0    0    0 S    0  0.0   0:00.06 migration/
    8 root      20   0     0    0    0 S    0  0.0   0:00.01 ksoftirqd/
    9 root      RT   0     0    0    0 S    0  0.0   0:00.16 migration/
   10 root      20   0     0    0    0 S    0  0.0   0:00.00 ksoftirqd/
```

Fig.7.CPU Utilization for FTP traffic



192.168.4.29 - PuTTY

```
top - 09:41:49 up 2 days,  5:10,  2 users,  load average: 3.59, 0.98, 0.33
Tasks:  81 total,   1 running,  80 sleeping,   0 stopped,   0 zombie
Cpu0  : 52.8%us,  1.0%sy,  0.0%ni,  0.0%id,  0.0%wa,  0.0%hi, 46.2%si,  0.0%st
Cpu1  : 84.7%us,  7.0%sy,  0.0%ni,  7.0%id,  0.0%wa,  0.0%hi,  1.3%si,  0.0%st
Cpu2  : 87.4%us,  9.0%sy,  0.0%ni,  0.7%id,  0.0%wa,  0.0%hi,  3.0%si,  0.0%st
Cpu3  : 91.0%us,  7.0%sy,  0.0%ni,  0.3%id,  0.0%wa,  0.0%hi,  1.7%si,  0.0%st
Cpu4  : 87.7%us,  5.0%sy,  0.0%ni,  6.0%id,  0.0%wa,  0.0%hi,  1.3%si,  0.0%st
Cpu5  : 74.8%us,  4.7%sy,  0.0%ni, 18.3%id,  0.0%wa,  0.0%hi,  2.3%si,  0.0%st
Mem:   8166756k total,  1325220k used,  6841536k free,    26388k buffers
Swap:        0k total,        0k used,        0k free,   179520k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 8914 root      20   0 1500m 1.0g 3128 S  561 13.1   4:32.38 Suricata-Main
   15 root      20   0     0    0    0 S    0  0.0   1:10.04 events/0
 9027 root      20   0  3120 1252 1028 R    0  0.0   0:00.45 top
    1 root      20   0  2496  764  660 S    0  0.0   0:05.42 init
    2 root      20   0     0    0    0 S    0  0.0   0:00.01 kthreadd
    3 root      RT   0     0    0    0 S    0  0.0   0:00.01 migration/0
    4 root      20   0     0    0    0 S    0  0.0   0:00.00 ksoftirqd/0
    5 root      RT   0     0    0    0 S    0  0.0   0:00.03 migration/1
```

Fig.8.CPU Utilization for HTTP traffic

4.3. Optimization of Suricata Configuration

Suricata IPS configurations are stored in a YAML file format[13].
Following options have been modified in the default yaml file[10] for optimizing the performance.

Max-Pending Packets:

Default number of packets allowed to be processed simultaneously by Suricata is 1,024. Increasing this limit to 5000 showed a slight improvement in performance.
max-pending-packets: 5000

Run-mode:

There are different run-modes available in Suricata. Workers mode is chosen since it gave better throughput than default auto-fp mode. In workers mode, all the treatment for a packet is made on a single thread.

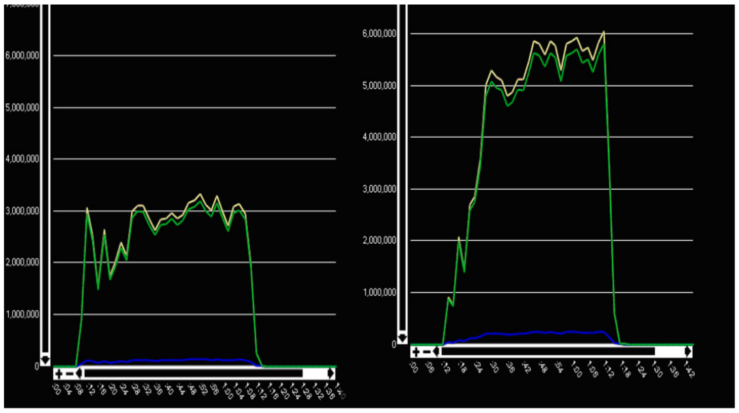
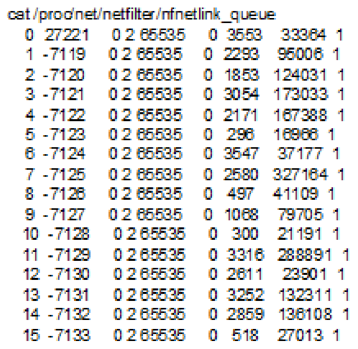


Fig.9.Graph showing throughput in default run-mode(autofp) versus workers mode of Suricata.

Netlink buffer size:

The default netlink buffer size in yaml is 18432. Increasing this value will increase the number of packets to be queued in nfqueue thereby reducing the packet loss. But when the value is changed to 20000 config was not getting reflected. Thus, the variable ―queue_maxlen‖ in the source code is explicitly set to 30,000.

Before this modification, there was significant packet loss.



```
cat /proc/net/netfilter/nfnetlink_queue
  0  27221    0 2 65535   0  3553   33364  1
  1  -7119    0 2 65535   0  2293   95006  1
  2  -7120    0 2 65535   0  1853  124031  1
  3  -7121    0 2 65535   0  3054  173033  1
  4  -7122    0 2 65535   0  2171  167388  1
  5  -7123    0 2 65535   0   296   16966  1
  6  -7124    0 2 65535   0  3547   37177  1
  7  -7125    0 2 65535   0  2580  327164  1
  8  -7126    0 2 65535   0   497   41109  1
  9  -7127    0 2 65535   0  1068   79705  1
 10  -7128    0 2 65535   0   300   21191  1
 11  -7129    0 2 65535   0  3316  288891  1
 12  -7130    0 2 65535   0  2611   23901  1
 13  -7131    0 2 65535   0  3252  132311  1
 14  -7132    0 2 65535   0  2859  136108  1
 15  -7133    0 2 65535   0   518   27013  1
```

Fig.9. nfqueue statistics before increasing queue length

Delayed-detect:

delayed-detect:yes
(default: it is commented) This option loads the signature after it starts capturing packets. Because of this suricata can capture the packets while signature building is in progress. This is used in DP suricata script to start/stop suricata without waiting for signature building.

Cpu-affinity:

set-cpu-affinity: yes
- receive-cpu-set: cpu: [―all‖] (default: cpu [0])
- decode-cpu-set: cpu: [―all‖] (default cpu[0, 1])

On dividing the cpu load among all the cores, performance is slightly increased.

In workers mode, number of threads depend on number of nfqueues used. If there are 16 queues used, there will be 16 worker threads in total.

Stream-reassembly:

stream: reassembly: memcap: 4gb #(default 128 mb)
The stream reassembly engine uses a set of pools in which preallocated segments are stored. There are various pools each with different packet sizes. Prealloc value for segments with different packet sizes are also modified for tuning the performance. (under segments: prealloc: in yaml file). This setting could be varied as per the verbose output.

Suricata rules:

By default, Suricata provides different rule files. For the deployment purpose, all the files containing rules are merged to all.rules (a single file).

5. METHODOLOGY

5.1. Selection of an open-source Intrusion Prevention System

Based on the literature survey done on various open-source Intrusion Prevention Systems such as Snort, Suricata and Metaflows, Suricata IPS is chosen for the deployment.

5.2. Cross-compilation

Cross-compilation is the act of compiling code for one system on a different system. The system on which compiler runs is the host and the system on which the new compiled program runs is the target. When host and target are of same machine types, the compiler is native. When host and target are of different machine types, the compiler is said to be cross-compiler. Since compilation is a resource-intensive process, it is difficult to do all resource-intensive tasks on embedded hardware designed for low-power consumption. Suricata programs are developed on X86 hardware. The processor on which Suricata IPS is deployed is MIPS based platform. Hence, Suricata is cross-compiled using MIPS tool chain on X86 machine that generated code for MIPS platform. Cross-compilation of Suricata involves crosscompilation of each library it requires and the cross-compilation of Suricata source code. Suricata requires following libraries: libpcap, libcapng, pcre, yaml, libnfnetlink, libnetfilter_queue, libmnl, libnet, libmagic, zlib, libnetfilter_log. All these libraries are downloaded and cross-compiled and finally the Suricata source too. The compiled binaries are then copied to proper the locations of processor where Suricata is being executed.

5.3. Plugging IPS into the packet path

Suricata IPS is a userspace software. In order to make the network packets to pass through Suricata userspace from kernel space[9] before it reaches the destination, IP tables rules have to be configured accordingly[3].

The simplest rule to send all traffic to Suricata is as follows:

```
iptables -I FORWARD -j NFQUEUE --queue-num 0
```

In this case, all forwarded traffic goes to Suricata through NFQUEUE. NFQUEUE is iptables and ip6tables[6],[7] target that delegates the choice on packets to a user space package. Once a packet reaches associate degree NFQUEUE[8] target it's en-queued to the queue as per the amount given by the --queue-num choice. The packet queue is enforced as a in chains list with component being the packet and data (a UNIX kernel skb). The protocol used between kernel and userspace is nfnetlink.

This can be a message primarily based protocol that doesn't involve any shared memory. Once a packet is en-queued, the kernel sends a nfnetlink formatted message containing packet knowledge and connected data to a socket and userspace reads this message. In userspace, the package should use libnetfilter_queue to attach to queue zero (the default one) and acquire the messages from kernel. It then should issue a finding of fact on the packet. To issue a finding of fact, userspace format a nfnetlink message containing the index of the packet and send it to the communication socket. As an example, the higher than rule can arouse a choice to a listening userpsace program for all packets aiming to the box.

--queue-balance is an NFQUEUE[9] option which to load balance packets queued by the same iptables rules to multiple queues. The usage is fairly simple. For example, to load balance FORWARD traffic to queue 0 to 15, the following rule can be used[12].

```
iptables -A INPUT -j NFQUEUE --queue-balance 0:15
```

The following command is used to view nfqueue statistics:

```
    cat /proc/net/netfilter/nfnetlink_queue

0 15015 0 2 65535 0 0 0 1
```

Col 1: queue num
Col 2: id attached to queue
Col 3: number of packets waiting to be processed by the application
Col 4: if packet payload is also passed, value is 2; if only meta-data is passed, value is 1
Col 5: how many bytes of packet payload should be copied to userspace at most.
Col 6: Packets dropped by kernel
Col 7: packets dropped within netlink subsystem
Col 8: ID of the most recent packet queued by userspace
Col 9: Always 1

5.4. Iptables rule setup to use IPS within the deployment context

Packets are being bridged here. So iptables are called in link layer forwarding context. The packets will go through ebtables NAT and then bridge forwarding action. Bridge forwarding action will call ebtables-forward-chain - filter table followed by iptables-forward-chain - filter table. The iptables forward chain has default DROP policy. Only all the accepted packets from forward chain are sent to Suricata to prevent unnecessary load. A new chain called IPS chain is thus introduced. The ACCEPT rules in FORWARD chain are replaced by a GOTO IPS chain target action. Also another rule with -J ACCEPT at the bottom is added in IPS chain. If IPS is enabled, NFQUEUE rule is added to IPS chain. If IPS is disabled, NfQUEUE rule is flushed and all the packets which hit allow action in FORWARD chain enter IPS chain and hit –j ACEEPT rule by which the packet is simply accepted.

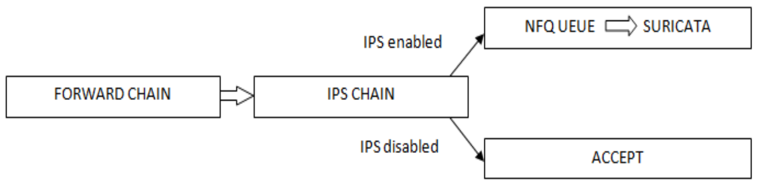


Fig 5.4.1: Packet flow through different chains of iptables

5.5. Deployment of IPS in data-plane part

On data-plane, all the libraries are copied and the Suricata binary is copied during build process. Iptables configuration are copied to a script and executed during init time. Along with this, ftpserver is started during init on data-plane. A script on control-plane passes the tar of Suricata configuration files using ftpput utility and invokes the script on data-plane to start Suricata binary. The script on data-plane copy the configuration files required for starting Suricata to proper locations, enable iptables rules with NFQUEUE option and start Suricata.

6. Conclusion And Future Work

Banking and Financial Organizations need security solutions to protect their data servers. Les deployed with LAN security solutions such as user authentication, validation of asset ownership, MAC address verification, NAC, firewall, traffic anomaly keep networks safer from network based attacks. These solutions equip the BFSI networks for the access control, transparency, visibility and the defense against malicious attacks which are the basic requirements of modern enterprise networks.

IPS feature deployed enhance the security features of a LE security system and prevent from suspicious threats.

The multi-core processor used here supports hardware acceleration. In this work, the IPS used has a software- based pattern matching engine. Future work could be porting the software based pattern matching engine to hardware-based pattern matching engine. Using hardware acceleration for pattern matching will increase the performance and give better results.

References

[1] Suricata Features, http://suricata-ids.org/features/all-features/

[2] A performance analysis of snort and suricata network intrusion detection and prevention engines. IDCS 2011, the Fifth International Conference on Digital Society, Gosier, Guadeloupe, France. 187–192.

[3] Deployment of Intrusion Prevention System based on Software Defined Networking, 2013 15th IEEE International Conference on Communication Technology (ICCT)

[4] Metaflows and its features, http://www.metaflows.com/features/ids/

[5] Free and open source intrusion detection systems: A study, 2015 International Conference on Machine Learning and Cybernetics

[6] Fundamentals of Iptables, http://www.thegeekstuff.com/2011/01/IPTABLES-FUNDAMENTALS/

[7] Iptables, https://help.ubuntu.com/community/IptablesHowTo

[8] About Nfqueue, http://netfilter.org/projects/libnetfilter_queue/

[9] Packet path through Kernel, http://www.cs.wustl.edu/~jain/cse567-11/ftp/pkt_recp/index.html

[10] Suricata.yaml,https://redmine.openinfosecfoundation.org/projects/suricata/wiki/Suricatayaml

[11] Tcpreplay, http://tcpreplay.synfin.net/wiki/tcpreplay

[12] Usage of nfqueue, https://home.regit.org/netfilter-en/using-nfqueue-and-libnetfilter_queue/

[13] Setting up Suricata in inline mode, https://redmine.openinfosecfoundation.org/projects/suricata/wiki/Setting_up_IPSinline_for_Linux

[14] Ubuntu Installation steps for Suricata, https://redmine.openinfosecfoundation.org/projects/suricata/wiki/Ubuntu_Installation

[15] Tuning Suricata Inline IPS performance- discussion, https://lists.openinfosecfoundation.org/pipermail/oisf-users/2011-December/001141.html

[16] Patrick-patch for zero copy, http://home.regit.org/2011/08/patrick-mchardy-memory mappednetlink-and-nfnetlink_queue/

[17] Suricata as a bridging IPS (Setup),http://taosecurity.blogspot.in/2014/01/suricata-20beta2-as-ipson-ubuntu-1204.html

[18] Emerging-Threats Ruleset Download, https://rules.emergingthreats.net/open/suricata/rules/

[19] Suricata Threading, https://kaurikim.wordpress.com/2015/02/16/suricata-threading/

[20] Tomahawk,http://tomahawk.sourceforge.net/