Global Information Assurance Certification Paper

Open Source IDS High Performance Shootout

GIAC (GCIA) Gold Certification

Author: George Khalil, George@GeorgeKhalil.com
Advisor: Rob VandenBrink

Accepted:
2-2-2015

Abstract

For many years, Snort has been the de facto open-source IDS/IPS solution, with the program's architects focused on improving the performance of this single-thread model. However, the demand for high-throughput IDS has increased as the average network throughput has increased from 1 Gbps to 10 Gbps, with 40 Gbps slated to become mainstream in the near future. In response, several other open-source projects have adopted a multi-threading approach to scaling IDS performance to meet the increasing demand for multi-gigabit analysis. As a result, Bro and Suricata are now viable candidates to replace Snort and are attempting to fill in the multi-threading gap left by Snort while leveraging existing Snort rule sets and third-party tools. Suricata and Bro have also introduced new features that were not originally explored by Snort, such as GeoIP lookups. Suricata is attempting to introduce GPU acceleration and IP reputation features to increase its throughput and compete with commercial IDS/IPS solutions that currently offer IP reputation functionality. To remain competitive, Snort has added a multi-instance feature to its 2.9 release to address its single-thread limitation and has indicated that version 3.0 will be multi-threaded by default. For a long time, Snort was the only player in the open-source IDS/IPS market, but now the industry is reaping the benefits of changing demands and increased competition through the introduction of new features and their integration with existing advantageous features.

1. Introduction

As early as 1972, the U.S. Air Force was becoming increasingly aware of computer security problems (Bruneau, 2001). The original focus of this awareness was the provision of shared computing resources while maintaining a safe, secure separation between classification domains—without compromising security. In 1980, James P. Anderson published a study introducing the use of accounting audits to detect incidents of unauthorized access (Bruneau, 2001). The U.S. government sponsored much of the IDS research conducted throughout the 1980s and 1990s in an effort to produce an automated real-time intrusion detection system (Bruneau, 2001). Several commercial IDS systems were developed offering varying capabilities. In 1998, Marty Roesch released the initial Snort open-source IDS, albeit with limited capabilities. In 1999, version 1.5 was released, introducing real-time packet analysis and logging capabilities. In 2000, Michael Davis imported Snort to Windows, bringing real-time IDS capabilities to the masses.

As network bandwidth increased, network-based IDS systems were challenged due to their single high-throughput choke points. A network-based IDS system funnels all network traffic through the sensor to detect anomalies. However, deep packet inspection requires significantly greater processing power. Prior to version 3.0, Snort's architecture relied on the efficient use of a single core. At the same time, network bandwidth increased from 100 Mbps to 1 Gbps and then 10 Gbps, and is now approaching 40 Gbps. A total of 100 Gbps networks are projected to be cost effective for mass adoption within the next few years. Sensor CPUs are no longer able to keep up with the exponential growth in network throughput. Currently, IDS sensor architecture attempts to reduce the amount of data processing required from a single sensor by distributing the processing across multiple CPUs or remote sensors. The volume of alerts generated by a high volume of traffic is also a challenge; for example, a single port scan of DNS services on Class B subnet would generate 65,000 alarms, overwhelming the analyst's console (Bruneau, 2001).

Additional concepts have evolved to address the issue of network congestion. Security architects pursued a distributed architecture by using host-based intrusion

detection. Each host in this architecture would be responsible for the analysis of its network traffic, while providing the additional benefits of an operating system and memory tampering detection. The distributed architecture provides a more scalable approach, as clients' processing power will continue to grow to address their analytical and detection needs. The large-scale, host-based IDS deployments are significantly scalable; however, the in-depth client interaction they require presents challenges that do not exist in network-based IDS systems. System patches, updates, and maintenance modify system behavior and have the potential to generate false positives and possibly trigger an IPS's automatic denial response. The volume of alerts generated by broad user interaction with systems requires an appropriately sized and trained team to deploy, maintain, and respond to events generated by the system (Chee, 2008).

Several other concepts have evolved based on anomaly detection and misuse detection. For example, network-based anomaly detection systems are typically installed in learning mode to define a standard operations baseline. Once activated, the system compares its baseline compared to traffic patterns and alerts the user if traffic exceeds the baseline (Debar, n.d.). Misuse detection IDS systems review the content of audit logs to identify normal behavior and flag abnormal behavior. Misuse-based systems parse system, network, or database logs to formulate patterns of normal behavior and identify and alert the user to abnormal behavior (Christina Yip Chung, n.d.). In the following sections, the author will explore the new features Snort, Suricata, and Bro are offering to the open-source community, as well as their varied approaches to accommodating 10-Gbps networks and beyond.

Snort, the most popular open-source IDS platform since its introduction in 1998, has no native packet capture facility; it requires an external packet-sniffing library (Koziol, 2003). Operating systems' network stacks typically re-assemble packets and provide application visibility to the packet's payload. A packet sniffer, on the other hand, requires access to raw unmodified packets to identify and detect anomalies. Snort is most commonly deployed using a LibPcap library as its packet-sniffing library of choice while providing support for other packet acquisition methods. LibPcap is widely supported across various operating systems, providing Snort with operating system flexibility.

Although LibPcap is versatile and has wide industry adoption, it's not a very efficient packet capture engine for high throughput systems. Libpcap can only process individual packets, making it a bottleneck for high-bandwidth monitoring (Koziol, 2003). Once the raw packets are delivered to Snort by Libpcap, Snort processes the packets using a series of decoders that are unique to each protocol element working its way up the protocol stack. Once the packets in a data flow are decoded, they move up to the preprocessor and detection engines for analysis. The preprocessor engine examines packets for suspicious activity and modifies them so that the detection engine can correctly interpret them by normalizing the traffic (Koziol, 2003). The traffic is cycled through every preprocessor to allow the identification of more sophisticated or obfuscated attacks. Snort offers a wide variety of preprocessors. The depth of protocol analysis and the configuration of the individual preprocessors have a direct impact on the amount of decoding required, as well as on IDS performance. The detection engine has rules to perform parsing and signature detection. The rules are parsed then loaded into memory for faster processing. Snort's detection engine uses a first match then exit logic. Once a packet matches a signature, Snort moves on to analyze the next packet. Figure 1 illustrates the packet and data flow as it enters and is processed by Snort.

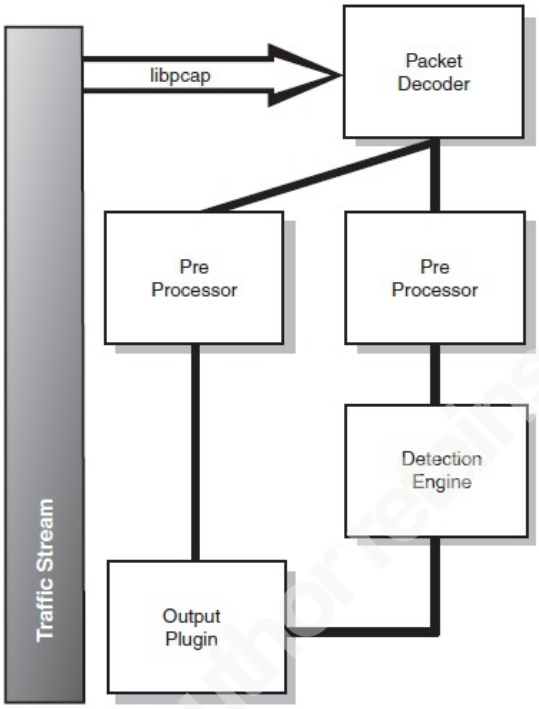

Figure 1 (Koziol, 2003)

Once Snort's detection engine or preprocessors identify malicious traffic, it is then sent to the output plugin. The output plugin offers various alert options, with the more detailed and concurrent output options requiring additional CPU cycles.

Suricata introduced features that were previously only offered by commercial IDS/IPS products and was designed to leverage all of Snort's existing community signatures and supporting tools. Data acquisition in Suricata is similar to Snort in supporting multiple-packet capture engines along with emerging signatures, as well as Snort's native signatures. Suricata's first beta release was in 2009, with the first standard release occurring in 2010 through the Open Information Security Foundation. The project was funded through grants from the Navy's Space and Naval Warfare Systems Command and the Department of Homeland Security's Homeland Open Security Technology. The Open Information Security Foundation is made up of "a multi-national group of leading software developers in the security industry." The foundation's primary goal is to identify current and future IDS/IPS needs and desires (openinfosecfoundation.org, 2014).

Suricata follows a similar data flow to Snort in acquiring packets by

defragmenting them, reassembling streams and normalizing application layer data as well as outputs for alerting and logging (To Linux and Beyond, 2013). The system is designed with native multi-threading, allowing multiple-packet capture queues to each worker process that spreads the workload across multiple CPU cores. The approach was implemented with the idea of spreading the processing across multiple processors to accommodate the increasing network throughput (Figure 2).

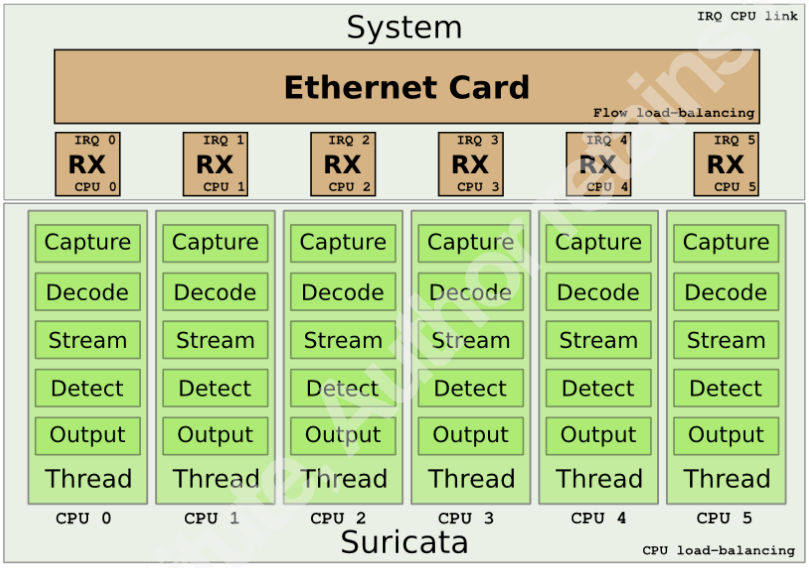

Figure 2 (bilgiguvenligi.gov, 2014)

Suricata also introduced some features that were not natively built in Snort. Alert and event filtering were added to newer versions of Suricata offering rate limits and thresholds per the host or subnet. IP reputation was also added to block known bad IPs, based on reputation reporting, to match some of the commercial IDS/IPS features. Graphics Processing Units (GPUs) offer significant multi-threading capabilities far exceeding that of CPUs. Password cracking and Bitcoin mining have been taking advantage of GPUs for several years. Suricata added CUDA GPU acceleration for pattern matching to further boost its systems' processing speed and distribute their workloads away from valuable CPU resources (Suricata-IDS.org, 2014).

Bro was developed to target scientific environments, network-savvy users, and the

Unix script driven Mindset (Sommer, 2011) . Bro supports Libpcap packet capture engines, which provide traffic to Bro's event engine. The event engine proceeds with protocol decoding and normalization then sends the data to the policy script interpreter for analysis logic. The policy script interpreter sends output to logs or notifications based on configuration parameters. Bro's development started in 1995 by Vern Paxson in Network Research Group at Lawrence Berkley National Lab (Mehra, 2012). The program's developers attempted to scale the system to a high-throughput network via Bro clustering. Bro provided primitive communication between sensors to offer clustering capabilities and multiplex a high volume of traffic across multiple sensors (Sommer, 2011). Bro was initially developed as a research tool; the initial focus was not on GUIs, usability, or ease of installation, and the program had the steepest learning curve versus Snort and Suricata due to its complexity (Schreiber, 2014). Bro differs from Snort and Suricata in its event-driven analysis versus the signature or anomaly-based detection offered in other IDS. Bro-Script is Bro's own policy engine, which provides analysis capabilities to download files on the wire, submit them to malware analysis, and notify the administrator. The event engine extends its power into possibly blacklisting the source and shutting down the user's computer that downloaded it through its power scripting and event-driven analysis (Schreiber, 2014).

2. Feature, Performance Comparison, and Optimization

2.1. Snort

Snort's architecture prior to version 3.0 is a highly efficient single-threaded analysis engine. Packet acquisition is the first step of traffic analysis; Snort supports a wide variety of packet capture engines. Snort's basic architecture uses the Libpcap packet capture library (opentodo.net, 2012). Libpcap is not optimized for high-throughput packet flow due to its serial packet processing (Koziol, 2003); Libpcap 0.9.x is the default capture framework on Linux for Snort 2.8.x and prior versions. Libpcap lacks built-in load balancing and is limited to a few hundred Mbits/sec of traffic. Snort 2.9 changed the program's default capture framework to AFPacket, increasing its throughput to 200-

500Mbits/sec, while still being limited to a single thread with no built-in load balancing. Other alternatives, such as Libpcap 1.0, were introduced, using features such as Mmap to improve performance, but had performance limitations due to hard-coded small buffer size. Mmap provides Libpcap access to a POSIX Unix system call to map files or devices directly into memory. PFRing is another high-throughput Linux kernel module that provides load balancing through a ring cluster design. PFRing also supports capture cards using a TNAPI/DNS high-performance driver. Using PFRing in conjunction with a Threaded New API (TNAPI) compatible NIC, Snort multi-instance configurations spanning multiple CPUs have been able to scale to 10 gigs per second on appropriately sized hardware. Custom hardware is available from SourceFire to simplify a multi-Snort deployment, as well as third-party vendors' custom-developed network interface cards, which provide load balancing features ranging in cost from $2,000 to $25,000 (Lococo, 2011).

CPU inspection is another significant factor in the function of high-performance IDS/IPS. It is estimated that Snort's CPU usage for each phase is approximately 10 percent for parsing, 10 to 20 percent for normalization and 70 to 80 percent for payload inspection. If the packets exceed the available CPU resources, packets will be dropped (Martin, 2011). The number of inspected rules results in an increase in processing requirements. Martin estimates that 1 CPU with 1,000 signatures is capable of examining 500 Mbps of network traffic. Martin produced a calculator based on performance monitoring of Snort's components, number of signatures, and network throughput. The calculator estimated that a Snort sensor requires 2.4 CPUs to support 4,000 signatures and analyse 400 Mbps of network traffic assuming 80 to 90 percent web traffic, as illustrated in the formula.

((4000/1000) = 4) * ((300/500) = .6) = 2.4 CPUs

By default, Snort does not support load balancing across multiple CPUs. Instead, the default configuration drops packets exceeding the capacity of a single CPU. Snort multi-instance feature was introduced to address this issue by launching a new Snort instance across each core.

Suricata has native multi-threading, therefore it normalizes the network traffic only once

prior to sending it to each worker thread for payload inspection. Snort, on the other hand, does not natively multi-thread and each Snort instance independently handles traffic normalization, incurring an estimated 10 to 20 percent CPU penalty per Snort instance. Advanced Snort deployments in high-throughput environments use dedicated hardware to handle the load balancing, such as PF_Ring and PCAP acceleration cards, or accomplish balancing through an external pcap load balancer. Offloading the packet analysis to hardware reduces Snort's CPU penalty down to 1 to 2 percent, freeing more CPU cycles for payload analysis (Martin, 2011). Performance testing by Eugene Albin and Neil Rowe at the U.S. Naval Postgraduate School identified Snort's optimal performance on commodity hardware to be limited to 200 to 300 Mbps per core (Rowe, 2011). Some vendors attempted to build intelligent cards with pcap offloading and on-card filtering capabilities so that only relevant traffic was passed to the sensor for payload analysis, such as cPacket's cTap (Deri, 2008).

2.2. Suricata

Suricata also supports a wide variety of packet acquisition methods, like Snort. The current Suricata release supports packet acquisition using the following capture methods:

- High-performance capture
  - AF_PACKET
  - PF_RING
- Standard capture
  - PCAP
  - NFLOG (Netfilter integration)
- IPS mode
  - Netfilter based on Linux (nfqueue)
  - ipfw based on FreeBSD and NetBSD
  - AF_PACKET based on Linux

•Capture cards and specialized devices

Endace

Napatech

Tilera (suricata-ids.org, 2014)

Suricata introduces new features that were not previously offered by Snort, but have long been available in other commercial products. Suricata provided the open-source IDS/IPS with new features, such as IP reputation, multi-threading, full IPv6 support, GeoIP lookup, and GPU CUDA acceleration for pattern matching (suricata-ids.org, 2014). Snort developers argued that multi-threading is not necessary once the overhead of distributed computing is compared to the efficiencies built into Snort. Roesch questioned Suricata's benefit, stating: "They've produced a clone of Snort that performs worse at taxpayer's expense." Matt Olney addressed the multi-threaded benefit when he wrote: "Internal testing pitting Snort against Suricata with rules loaded, Suricata runs up to about 200 MB per second. Snort, with rules, hits 894 MB per second with no drops" (Gerber, 2010). Suricata's latest release significantly improved its performance per core. A network interface card can produce significant performance improvement by providing multiple buffer queues to CPUs, increasing the multi-threading performance. Vendors demonstrated 10-gigabit cards that can double Suricata's performance, such as the Emulex FastStack card introduced in Black Hat 2012 (Emulex, 2012).

The number of published Suricata performance papers is limited. The Naval Post Graduate School (NPS) published a performance benchmark comparing Snort 2.9.x to Suricata 1.0.x in 2011. The tests revealed that Snort single instance is more efficient, with nearly 50 percent less memory utilization than Suricata. The NPS author attributed the extra memory usage to the overhead associated with tracking multiple detection threads. Snort's efficiency is degraded in a multi-instance implementation. Suricata was tested on a 48 CPU hosting 12 cores per CPU and 125 GB of RAM and a network throughput of 20 Gbps. Using the available hardware at the time resulted in Snort dropping packets at the rate of 53 percent versus Suricata dropping 7 percent of the packets. The author also noted the fast development of Suricata at the time in relation to the stable Snort release

and its potential impact on the production environment (Albin, 2001). Multiple authors have indicated that the 2.0.x version of Suricata improved its performance per thread to Snort-equivalent levels. The author of this paper was unable to find any published performance benchmarks for the 2.0 release of Suricata without the use of specialty network cards or boards, however. Peter Manev did publish a guide to deploying a 10-Gbps Suricata version 2.0 dev (rev 92568c3) IDS/IPS. This implementation guide is based on a 64-bit Ubuntu LTS 12.04.2, Intel Xeon E5-2680, 64 gigs of RAM, and an Intel 82599EB 10-gigabit SFI/SFP+ network card. Manev's deployment uses the PF_Ring Direct NIC access, AF_Packet. The latter implementation was tested on a 10 Gbps network transmitting 9.7 Gbps per data for 13 hours with an impressive 1.897 percent packet loss rate (Manev, 2013).

Albin's research at the Navy's Post Graduate School measured the accuracy of Snort and Suricata's detection against a control PCAP file with known control events. Snort's earned an 81 percent, while. Suricata achieved 91 percent detection accuracy from the control PCAP file. The author suggested that common false negatives were due to the signature-related issues, rather than a detection engine problem, as both systems utilized the same signature rule set. Albin also noted that during his testing, Suricata's development team released three minor versions and two betas, while Snort had the same production release for five months (Albin, 2001).

Suricata introduced IDS/ GPU (Liberios Vokorokos1, 2012). Limited testing was conducted using the Nvidia GTX 260 216 IPS detection engine offloading as an option to reduce the CPU workload. Non-parallelizable rules were analyzed by the CPU and parallelizable rules were analyzed by the Core GPU. Suricata version 1.3 yielded a 21 percent reduction in processing time for a 20-MB PCAP control file (Liberios Vokorokos1, 2012). Suricata version 2.0 was tested using an Nvidia GTX 480 448 Core GPU as opposed to CPU utilizing various PCAP control files. The results were similar to those of previous tests, indicating a performance gain ranging from 0 percent and up to 30 percent when using GPU offloading (Poona.me, 2013). Research unrelated to IDS/IPS explored GPU's potential to exceed CPU significantly. GPU IDS research was published by Robert Ricci and Weibin Sun of the University of Utah's School of Computing.

Ricci's research proposed a SNAP-module-based router framework to utilize GPU's parallel computing power to process packets. GPU benchmarking yielded an impressive 30.97 Gbps in simple forwarding mode and an equally impressive 18.8 Gbps using SDN forwarder for classifying packets based on headers, GPU processing reaching up to 559 million packets per second (Weibin Sun, 2013). Suricata's GPU offloading is still being fine-tuned and has not yet achieved a performance comparable to that of SNAP.

Suricata also introduced a potential forensic tool to their IDS through the file extraction and logging feature. File extraction and logging has over 4,000 file types built in, as well as MD5 matching. The MD5 hash allows administrators to create powerful rules to deny pre-defined MD5 hashes access to their IPS. Known malware MD5 hashes can still be loaded as a part of the signatures and provide an automated IPS response if the file hash is detected, however. Suricata's MD5 file structure is limited to 32-bit systems, so the maximum theoretical MD5 list can reach 4 GB and allow approximately 250 million MD5 hashes. Inliniac tested a 300-Meg MD5 hash using Suricata version 1.3 beta2 and concluded that MD5 hash matching has no significant negative performance impact (Inliniac, 2012). Martin Holste tweeted the capability to negate the blacklist command and effectively use Suricata only to pass explicitly permitted files (Inliniac, 2012). Openinfosecfoundation.org authored a guide to implementing NIST's whitelist MD5 DVD into Suricata and applying it as a whitelist to the MD5 hash-matching engine (Manev, 2014). In addition to the whitelist/blacklist functionality, Suricata offers a hybrid option between the MD5 matching and Filestore or File-Keywords to store particular files for further analysis only if they match a pre-defined MD5 hash. File-Keywords offers a broad set of features to store specific file extensions, file names or file magic signatures.

2.3. Bro

Bro is a flexible script-driven intrusion detection system. Unlike Snort or Suricata, Bro does not offer inline intrusion prevention features; it offers new features through its script decision options to drop, sample, throttle, or redirect packets. The intrusion detection system was developed by Vern Paxon of the International Computer Science Institute in Berkeley, CA. The sensor supports commodity hardware on standard Linux, FreeBSD, and MacOS platforms. Bro's capabilities originate in academic research

projects. However, that research background increases the learning curve for users implementing Bro in production environments (Bro.org, 2014). Bro's internal architecture differs from its open-source alternatives in its script-driven policy engines. Rather than rely on separate processing engines, processors, and decoders, Bro relies on script interpreter. Packets are acquired from the network using standard packet acquisition libraries, such as libpcap. With the release of Bro version 2.3, PF_Ring ZC native support was introduced to achieve 1/10 Gbit line rate packet processing in both RX and TX (ntop.org, 2014). PF_Ring zero copy also offers integration of zero-copy functionality to vms and non-PF_Ring-aware devices, such as WiFi or Broadcom NIC's (ntop.org, 2014). In addition to utilizing high-throughput packet acquisition engines, Bro offers clustering options for high-throughput environments. Bro does not have built-in multi-threading capabilities to address high-throughput networks; Bro provided a "worker"-based architecture to utilize multiple processors (Figure 3).

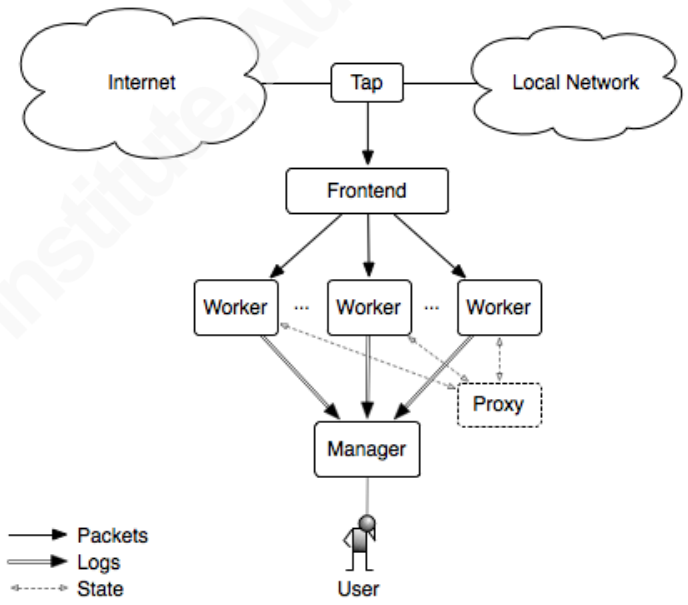

Figure 3 (Bro.org, 2014)

Bro also does not have a built-in front traffic-splitting option. Several vendors offer discrete hardware flow balancers. cPacket offers hardware solutions that perform layer-2 load balancing by rewriting the destination mac address to cause each packet associated with a particular flow to have the same destination MAC (Bro.org, 2014). This method

relies on cPacket's hardware to distribute 10G interface traffic across multiple 1G interfaces and associated worker processes. Each worker would then use a Berkeley packet filter to limit the inspection engine's visibility to only that stream of flows reducing the cost through the utilization of 1G interfaces (Bro.org, 2014). OpenFlow switches are software switches offering the potential to do flow-based balancing using FlowScale directly on the switch, rather than on the NIC or through third-party hardware (OpenFlowHub.org, 2012). PF_Ring offers clustering (ntop.org, 2014) on Linux by spreading the flow-based load balancing across a number of processes that are sniffing the same interface (Bro.org, 2014).

Bro has the option to configure multiple worker threads to receive the packet streams from the front end. Bro's developers recommend allocating one core for every 80 Mbps of traffic that is being analyzed (Bro.org, 2014). Worker processors handle all the protocol parsing and send logs to a remote manager. Bro's processing per core is significantly lower compared to Snort's 800 Mbps per processor and Suricata's starting rate of 200 Mbps per core. However, having the built-in capability to spread the load across multiple machines via Bro cluster provides greater scalability, as the code is tuned and performance improves over time. The manager process has two primary functions: to receive logs messages and notices from the rest of the nodes in the cluster and to deduplicate notices. The result of message and notice synchronization between the cluster nodes is a single log for the entire Bro deployment (Bro.org, 2014).

Although Bro offers worker processes and clustering to distribute a large network's traffic load across multiple CPU cores, as well as multiple servers, Snort's developers highlighted the overhead of distributed processing. Nicholas Weaver and Robin Sommer published a paper on the topic of stress testing the Bro cluster. The test cluster was set up using ten sensors; each sensor contained a dual CPU Xeon-based system with hyper-threading enabled. Traffic was generated using a 500x multiplication of the HTTP stream from a single transmitter. In testing a single processor, Weaver and Sommer found that a Bro instance was able to handle a 100-factor HTTP stream at 4,900 pps before it started to drop traffic (Sommer N. W., 2007). When the stream factor was increased by 500, Bro was only able to handle 3,600 pps. The test was repeated by

increasing the number of processing cluster nodes and the results are recorded in Table 1 and Table 2 below.

Number of Sensor Nodes	Maximum Processing Rate	Speedup
Standalone Bro	3600 pps	
2 Sensors	6800 pps	1.8×
4 Sensors	14400 pps	4.0×
8 Sensors	35400 pps	9.8×

Table 1: The Scalable Performance of Cluster Bro on the 500x multiplied HTTP trace

Number of Sensor Nodes	Multiplication Factor	Maximum Processing Rate	Speedup
Standalone Bro	400	3800 pps	
2 Sensors	800	6600 pps	1.7×
4 Sensor	1600	13100 pps	3.4×
8 Sensors	3200	25500 pps	6.7×
16 Sensors	3200	55700 pps	14.7×
24 Sensors	3200	88400 pps	23.2×
31 Sensors	3200	84400 pps	22.2×

Table 2: The performance of Cluster Bro when the number of streams is scaled with the size of the cluster.

Table 1 and Table 2 (Sommer N. W., 2007)

The Bro cluster research confirmed that distributed computing overhead is only efficent up to a certain saturation point. In the case of Weaver and Sommer's research, the saturation point appears to be near the 31-cluster node count. However, this limit can be improved, as the software is optimized for distributed computing as processing and thread tracking efficiency is improved.

Bro script offers significant flexibility in detection options, logging, and post-detection action. Snort and Suricata's in-line approach provides options to log and block traffic that matches the signature. Bro's extensible scripting provides opportunities to interact with other systems in the enterprise. In addition to logging and blocking, Bro offers options to send email messages, page on-call staff, or automatically terminate existing connections (Gerber, 2010). Bro offers unique options for rate-limit flows that match a configured policy, as well as the operating system and application response through the scripting policy. Bro's matching policy differs from that of Snort and Suricata due to its context-aware framework. Snort and Suricata use the Snort rule base; Bro previously offered Snort2Bro script to convert Snort-compatible rules to the Bro format. Due to the significant differences in matching signatures to the environment and user-

configured context, Bro developers removed that script in version 2.3.1. Bro 2.2 introduced file extraction and matching features similar to those of Suricata. File hash extraction support and matching permits automated file extraction and alerts using custom file hashes or through publicly available hash registries, such as Team Cymru's Malware Hash Registry (Bro.org, 2014).

Although the cluster stress testing research is several years old, it has been commonly acknowledged that the overhead increases as the workload is distributed across additional processing nodes (Balderrama, Huu, & Montagnat, 2012). A focus on improving per-core processing efficiency is needed; however, the need to increase efficiency in a clustered or distributed environment is critical to keeping up with the sustained growth in network throughput.

3. Conclusion

An IDS/IPS drops packets when it's processing limits are reaches. A single thread or single processesor is unable to keep up with 10/40 Gig throughput.. The resulting packet loss due to a saturated IDS/IPS provides increased potential for false negatives. As the percentage of unanalyzed traffic increases, the statistical likelihood of missing events of interest also increases. IDS/IPS rely heavily on computer processing power to perform deep packet inspection and pattern matching. Moore's Law predicted that the number of transistors will double in capacity every 18 months, potentially doubling computing power within the same period. Manufacturers reached technological limitations in regard to reducing transistor sizes, although silicon makers were able to extend the limits of Moore's Law by scaling to multiple cores and multiple processors. However, it's been predicted that silicon makers will only be able to extend Moore's Law through 2015 and possibly through 2020 as they reach the lower economic limits of transistor size (Templeton, 2014). At the same time, Nielsen's Law of Internet Bandwidth predicted that a high-end user's connection speed grows by 50 percent per year (Nielsen, 2014). As silicon makers reach the maximum number of transistors per die and users' available bandwidth increases 50 percent annually, the need to offload processing across all available hardware, including processors, GPUs, and specialized network cards, also

increases.

Network throughput is growing at an exponential rate. Commercial IDS/IPS products are increasing their processing power through custom ASICs and proprietary hardware. The open-source community is seeking a scalable solution that utilizes commodity hardware. Snort's developer, Olney, described Snort's future Razorback framework as follows: "It isn't Snort, it isn't ClamAV, and it isn't Suricata. It's a new approach to the detection problem and was built from the ground up in close collaboration with groups that are facing APT-level threats. It may not be perfect; it may not even be the right answer (but we think it is), but it is truly innovative" (Gerber, 2010). The innovation and different approaches developed by the major open-source IDS/IPS groups are critical to solving high-volume analysis challenges. Future Snort versions will utilize a highly optimized multi-threaded engine. Suricata is already multi-threaded and is increasing in efficiency as it continues to mature. Bro is attempting to maximize its performance while natively scaling across multiple sensors. As 40- and 100-gigabit adoption increases, time will tell which approach is more efficient. Each IDS product offers a unique way to address high-throughput challenges. Snort is an extremely well-tuned, single-threaded product; Suricata leverages the Snort ruleset, as well as other supporting products, with the addition of multi-threading, as well as file extraction and hash white- and blacklisting. Bro offers additional features through its script-based analysis engine and capability to extend the response via scripts.

"Defense in depth is the coordinated use of multiple security countermeasures to protect the integrity of the information assets in an enterprise" (Rouse, 2014). Best practice is to layer security architecture through the use of multiple solutions. The Navy Post Graduate School testing validated that Snort produced more false negatives due to packet loss than Suricata. Bro also offers scripting features that cannot be utilized with Snort and Suricata. The author therefore recommends strategically applying defense in depth using open-source IDS/IPS products that constitute a mix of the three aforementioned products. Bro and Suricata can provide significant advantages when deployed in 10-Gbps and beyond networks. Open-source Snort can be deployed at the organization perimeter or between enclaves, where bandwidth is below 1 Gbps. Once

Snort 3.0 is released, higher throughput will be possible using Snort open-source products and commodity hardware. Even if all products can achieve the same speed, they all apply different inspection options, which can provide benefits within the same environment.

Each IDS has strengths and weaknesses that can assist in the selection of an appropriate solution that best fits each organization. Snort is a great fit for commercial organizations, where enterprise support options, along with a broad user base, are advantagous. ISPs and hosting providers using 10-gigabit links might find Suricata's multi-threading advantageous, as well. ISPs deploy commodity hardware and are open to the use of open-source software without the need for commercial support. Bro, on the other hand, was developed within universities and remains best suited for high-throughput research environments. The research-driven culture in universities, along with their plethora of available graduate students, provides the resources required to leverage the full power of Bro and its powerful scripting features. The differences between the three IDS products provide a perfect demonstration of how defense in depth can reduce risk by drawing upon multiple technologies to solve the same problem.

4. References

bilgiguvenligi.gov. (2014). Retrieved from https://www.bilgiguvenligi.gov.tr/images/stories/kucukparmak/uk/idsips/suricata-workers-af-packet-en.png

Albin, E. (2001, 09). *A COMPARATIVE ANALYSIS OF THE SNORT AND SURICATA INTRUSION-DETECTION SYSTEMS*. Retrieved from Naval Postgraduate School: http://faculty.nps.edu/ncrowe/oldstudents/ealbin_thesis_final.htm

Balderrama, J., Huu, T., & Montagnat, J. (2012, 06). *Scalable and Resilient Workflow Executions on Production Distributed Computing Infrastructures*. Retrieved from ieee.org: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6341502&isnumber=6341486

Bro.org. (2014, 12 06). *Bro 2.3.1 Documentation*. Retrieved from Bro Network Security Monitor: https://www.bro.org/sphinx/intro/index.html

Bro.org. (2014, 12 06). *Bro Cluster Architecture*. Retrieved from Bro.org: https://www.bro.org/sphinx/cluster/index.html

Bro.org. (2014, 12 07). *Bro File Analysis Exercises*. Retrieved from Bro.org: https://www.bro.org/bro-exchange-2013/exercises/faf.html

Bruneau, G. (2001). *The History and Evolution of Intrusion Detection*. Retrieved from SANS.org: http://www.google.com/url?sa=t&rct=j&q=&esrc=s&frm=1&source=web&cd=1&cad=rja&uact=8&ved=0

CB4QFjAA&url=http%3A%2F%2Fwww.sans.org%2Freading-room%2Fwhitepapers%2Fdetection%2Fhistory-evolution-intrusion-detection-344

Chee, J. (2008). *Host Intrusion Prevention Systems and Beyond*. Retrieved from SANS.ORG: http://www.sans.org/reading-room/whitepapers/intrusion/host-intrusion-prevention-systems-32824

Christina Yip Chung, M. G. (n.d.). *DEMIDS: A Misuse Detection System for Database Systems*. Retrieved from Department of Computer Science, University of California at Davis: http://www.csee.umbc.edu/courses/pub/cadip/docs/NetworkIntrusion/IFIP99.pdf

Debar, H. (n.d.). *Intrusion Detection FAQ: What is behavior-based intrusion detection?* Retrieved from SANS.org: http://www.sans.org/security-resources/idfaq/behavior_based.php

Deri, L. (2008, 04). *IP Traffic Monitoring at 10 Gbit and above*. Retrieved from terena.org: http://www.terena.org/activities/ngn-ws/ws2/deri-10g.pdf

Emulex. (2012). *Integrating and Optimizing Suricata with FastStack™ Sniffer10G™*. Retrieved from SlideShare.net: http://www.slideshare.net/emulex/integrating-and-optimizing-suricata-with-faststack-sniffer10g

Gerber, J. (2010, 08 26). *Three Open Source IDS/IPS Engines: The Setup*. Retrieved from Security Advancements at the Monastery: http://blog.securitymonks.com/

Inliniac. (2012, 09 06). *Suricata MD5 Blacklisting*. Retrieved from Inliniac, Everything inline: http://blog.inliniac.net/2012/06/09/suricata-md5-blacklisting/

Koziol, J. (2003). *Intrusion Detection with Snort.* Sams Publishing.

Liberios Vokorokos1, A. B. (2012). *Intrusion Detection Architecture Utilizing Graphics Processors*. Retrieved from Acta Informatica Pragensia: http://aip.vse.cz/index.php/aip/article/download/12/14

Lococo, M. (2011, 08 04). *Capacity Planning for Snort IDS*. Retrieved from mikelococo.com: http://mikelococo.com/2011/08/snort-capacity-planning/

Manev, P. (2013, 12 08). *Suricata (and the grand slam of) Open Source IDPS* . Retrieved from IT Security through Open Source : http://pevma.blogspot.se/2013/12/suricata-and-grand-slam-of-open-source_8.html

Manev, P. (2014). *Filemd5 and white or black listing with MD5 hashes*. Retrieved from openinfosecfoundation.org: https://redmine.openinfosecfoundation.org/projects/suricata/wiki/Filemd5_and_whiteblack_listing_with_MD5

Martin. (2011, 04 11). *Network Intrusion Detection Systems*. Retrieved from Open-Source Security Tools: http://ossectools.blogspot.com/2011/04/network-intrusion-detection-systems.html

Mehra, P. (2012). A brief study and comparison of Snort and Bro Open Source Network Intrusion Detection System. *International Journal of Advanced*

Research in Computer and Communication Engineering Vol. 1, Issue 6, August 2012, 384.

Nielsen, J. (2014). *Nielsen's law of internet bandwidth*. Retrieved from Nielsen Norman Group: http://www.nngroup.com/articles/law-of-bandwidth/

ntop.org. (2014, 04). *PF_Ring User Guide*. Retrieved from ntop.org: https://svn.ntop.org/svn/ntop/trunk/PF_RING/doc/UsersGuide.pdf

ntop.org. (2014, 12 06). *PF_Ring ZC (Zero Copy)*. Retrieved from ntop.org: http://www.ntop.org/products/pf_ring/pf_ring-zc-zero-copy/

OpenFlowHub.org. (2012, 11 14). *FlowScale*. Retrieved from OpenFlowHub.org: http://www.openflowhub.org/display/FlowScale/FlowScale+Home

openinfosecfoundation.org. (2014, 11 11). *openinfosecfoundation.org*. Retrieved from openinfosecfoundation.org: http://www.openinfosecfoundation.org/

opentodo.net. (2012, 10 17). *Snort From Scratch Part I*. Retrieved from opentodo.net: http://opentodo.net/2012/10/snort-from-scratch-part-i/

Poona.me. (2013, 06 21). *Suricata cuda engine re-designed*. Retrieved from Poona.me: http://www.poona.me/2013/06/suricata-cuda-engine-re-designed.html#performance

Rouse, M. (2014, 12 14). *Defense in Depth*. Retrieved from techtarget.com: http://searchsecurity.techtarget.com/definition/defense-in-depth

Rowe, E. A. (2011). *A Realistic Experimental Comparison of the* . Retrieved from Navy PostGraduate School: http://faculty.nps.edu/ncrowe/suricataeval_fina12.htm

Schreiber, J. (2014, 01 13). *Open Source Intrusion Detection Tools: A Quick Overview*. Retrieved from AlienVault.com: https://www.alienvault.com/blogs/security-essentials/open-source-intrusion-detection-tools-a-quick-overview

Sommer, N. W. (2007, 12 07). *Stree Testing Cluster Bro*. Retrieved from usenix.org: https://www.usenix.org/legacy/event/deter07/tech/full_papers/weaver/weaver.pdf

Sommer, R. (2011). *Bro IDS The open Source Bro IDS - Overview and recent development*. Retrieved from ICIR.org: http://www.icir.org/robin/slides/Bro-CACR-Indianapolis.pdf

suricata-ids.org. (2014, 11 26). *Complete list of Suricata Features*. Retrieved from Suricata Open Source IDS/IPS/NSM Engine: http://suricata-ids.org/features/all-features/

Suricata-IDS.org. (2014, 11 15). *Suricata All Features*. Retrieved from suricata-ids.org: http://suricata-ids.org/features/all-features/

Templeton, G. (2014, 03 25). *The end of Moore's Law may already be here*. Retrieved from Geek.com: http://www.geek.com/chips/the-end-of-moores-law-may-already-be-here-1588927/

To Linux and Beyond. (2013). *Flow Reconstruction and normalization in Suricata*. Retrieved from To Linux and Beyond: https://home.regit.org/2012/11/suricata-flow-reconstruction/

Weibin Sun, R. R. (2013). *Fast and Flexible: Parallel Packet Processing with GPUs and Click*. Retrieved from ACM Digital Library: http://dl.acm.org/citation.cfm?id=2537861

George Khalil, George@GeorgeKhalil.com