This is a GIAC Gold Template

Intrusion Detection and Prevention Systems

A virtual look at a physical technology

GIAC (GCIA) Gold Certification

Author: Christopher Hoke, chris.hoke@continuumww.com
Advisor: Tim Proffitt

Accepted: November 16th 2012

Abstract

This paper will explore the feasibility of combining open-source network and host-based monitoring tools to build a virtualized appliance capable of both detecting common attacks and serving as an at-rest data loss prevention platform on a host operating system. The research will explore the performance and functionality impacts of using common open source tools such as Snort, Suricata, OpenDLP, VMware Player, and open-source operating systems to accomplish goals that would otherwise require significant capital outlay to achieve.

1.0 Introduction

Defending connected networks has been a challenge for as long as there have been connected networks. Well-known examples of successful attacks against computer systems include the Morris Worm and the compromise documented in *The Cuckoo' s Egg* in the 1980' s up to the LulzSec and Anonymous attacks of the 2010' s. Throughout, the concept of Network Security Monitoring, or NSM, has gained popularity as a holistic approach to network defense. At its core, NSM focuses on monitoring network traffic as it traverses the network.

The tools used to process the monitored traffic provide context for human review and analysis. This paper will focus on two of these technologies. Intrusion prevention systems, or IPS, are systems intended to provide active detection and protection against an attack or intrusion from outside of a firm' s network over the Internet. In the recent past, the technologies that compose intrusion detection systems and intrusion detection systems have largely converged. For example, Snort (a technology discussed in further detail later) now has the ability to act as an IDS or an IPS through incorporation of active response. Thus, in this paper, the term IPS will be used to identify the technology as a whole, except where contrast is needed.

The second, Data Loss Prevention, or DLP, focuses its detection and prevention capabilities on identifying data leaving a network. When DLP is brought in to an organization, data is identified (by policy or classification) that should not be allowed to leave the network. (Mogull, 2010) DLP tools utilize a combination of installed agents and traffic proxies to detect and prevent unauthorized movement of identified data.

The purpose of this project is to discuss how IPS and DLP work, how they differ from other detection systems, and the strengths and weaknesses of these systems. This project is also interested in looking at the feasibility of creating a virtual appliance that could be used as a virtualized host based IPS/DLP device. This device would be a low-cost option for organizations that need coverage, but don' t have the capital resources to purchase enterprise-grade technology. These goals will be accomplished through the discussion of general IPS theory, DLP theory, and host-based systems.

2.0 Network Detection and Prevention Technologies

2.1 Intrusion Prevention Systems

IPS came about as a natural evolution of IDS. The majority of traditional IDS used a static signature base to detect malicious traffic. This meant that the system was only able to detect attack traffic that was already known to it. If there was not a signature for a particular attack or exploit, such as zero day attacks, the IDS was not able to detect or report it. Also, IDS were generally not placed in-line; they were able to alert on malicious traffic, but had no ability to prevent that traffic. In addition, because IDS were passive and based on signatures, they were only as good as the people responsible for running them. As a result, while most IDS products had an effective rule set out of the box, without constant attention and tuning the IDS soon became ineffective. A poorly tuned sensor could generate up to a 95% false positive rate. (Northcutt and Novak, 2002)

IPS were created to improve on some of the weaknesses of IDS. There are three main types of IPS, those that use generic signatures, those that are application or host-based, and deceptive applications. (Desai, 2003) Gartner identified three requirements a good IPS needs to meet in order to be effective and appropriate in any organization. First, the IPS must not disrupt normal operations. After it is placed in line, it cannot cause lag or any excessive use of resources. Second, they must block attacks using multiple algorithms. In short, while a strong signature base is important, an IPS must also include some adaptive behavior and anomaly detecting algorithms. Last, an IPS must be able to distinguish a true attack from normal traffic. That is, it should be able to minimize false positives and false negatives. (MacDonald, 2006)

IPS that use generic signatures are systems that sit at the network level and use a generic rule set to detect malicious traffic. Examples of this type are in-line IPS and layer seven switches. These types of IPS are able to detect malicious traffic as it passes and either send a message to the firewall or initiate blocking themselves where appropriate. While these are not much more than a “beefed-up” traditional IDS, they can be effective. However, because they are IDS based, they are still only able to detect traffic they have rules for. This type of device will not be able to stop zero day exploits or protocols it doesn’t recognize. These systems are good for defending systems like mainframes that are otherwise difficult to protect. (Desai, 2003)

Application or host-based IPS are able to detect malicious traffic through interaction with the protected system. The IPS is able to view “normal” traffic for the host or application and learn what traffic is legitimate and which traffic is malicious. Application-based IPS function on a “allow by default” basis. If an application-based IPS does not recognize traffic that is attempting to pass through, it will not allow the traffic to pass. Host-based IPS will be discussed in greater depth later in this paper.

The third type of IPS are called deceptive applications. These are applications that not only identify and block malicious traffic but also are also able to interact with the attacker in order to gain information that can be used to identify and permanently block their address. These IPS work by sending specially crafted packets in response to the attacker. This not only protects a firm’s network; it also can give insight into an attacker’s methods and choice of exploits. (Desai, 2003)

Some strengths of IPS systems are that they generally have a lower instance of false negatives than traditional IDS systems. A false negative is a packet that should have been alerted on by the system, but is recognized as acceptable traffic. Another strength is that most IPS systems focus on allowing acceptable traffic rather than on blocking bad traffic. As a result, the rule sets tend to be stable and require less maintenance and updating. In addition, an IPS provides an analyst with the ability to actively respond to detected bad traffic. While this is a powerful tool, it should not be used indiscriminately. It can be effective, but if bad traffic is being spoofed to a third party address it can cause harm to an innocent party. (Northcutt and Novak, 2003) A final strength of IPS is that they generally require less analyst intervention. A properly configured IPS should reduce the number of false positives generated. Combined with the fact that IPS should require less updating and maintenance, it is clear that analysts will have to spend less time with the IPS.

IPS are not without their weaknesses. These systems can require increased system resources due to the increased CPU time required to perform deep packet analysis and active response. (Paulson, 2002) Also, a poorly configured IPS can have as many false positives as traditional IDS; if active response is enabled, false positives can cause blocking or session sniping of acceptable traffic. (Northcutt and Novak, 2003)

2.1.1 Snort

Snort is an open-source, free IDS system. Snort runs on multiple platforms including Linux, Unix, and Windows. Unlike many commercial solutions, Snort is completely transparent; users can see and configure everything from the rule set to the underlying source code. (Caswell & Hewlett, 2007) Snort has traditionally been used as a network intrusion detection system. That is, Snort sensors are typically placed in area of high traffic often connected to the spanning port on a switch, or to a tap. This placement allows a single Snort sensor to see all of the traffic on a given network or segment.

Snort considers itself to be "behavior based" rather than rules or signature based. While each of these does compare sniffed packets against a rule set, a signature based tool needs to match a particular string or sequence of characters in a packet. These types of sensors are vulnerable to a change in the packet content, similar to the difficulties antivirus companies face in responding to emerging viruses. When writing for behavior based sensors, the rule is written for the vulnerability, not the exploit. For example, if it is known that a particular vulnerability can be exploited by a packet containing data that starts at a particular byte within that packet and that a successful exploit of the vulnerability always shows a hex value of |FF| 3 bytes after the starting byte, a rule can be crafted to match this behavior rather than the content of the payload; by checking for the presence of these two offsets, Snort can detect the exploit regardless of what else may be in the payload (Caswell & Hewlett, 2007)

The power of the Snort rule base is the biggest strength of Snort, but is also its most glaring weakness. Because of the complexity of the rule base Snort can create lots of noise on a network. False positives are a trademark of traditional IDS technology in general and Snort is no exception. In order for a Snort implementation to be successful, there has to be an effort to tune the sensors and rules.

According to Sourcefire, Snort is the most widely downloaded IDS ever. Its widespread adoption is due to the fact that it is open-source and free to download and deploy. The Snort rules are also available from Sourcefire free of charge as well, and are updated monthly. Unfortunately, in this case open-source and free means no support, and Snort is a complex tool to deploy and implement from scratch. While Snort is free, the number of quality free tools for centralized sensor and rules management is limited. The result is that the majority of Snort configuration has to be done from the command line on each machine Snort is deployed on. Additionally, because Snort is cross-platform, it can be difficult to identify OS specific dependencies and idiosyncrasies needed to make Snort work. For example, in working on this project, the author found that CentOS was much easier to work with than OpenSolaris or OpenBSD. There can be tradeoffs however, as either of these could be a more secure operating system out of the box.

2.1.2 Suricata

Similar to Snort, Suricata is also an open-source software product. According to the Open Information Security Foundation, Suricata is an attempt to “build a next generation IDS/IPS engine”. While Suricata is able to leverage the traditional syntax of Snort rules, it takes advantage of advances in modern hardware. According to its creators Suricata is a “multi-threaded engine” able to leverage both hardware acceleration and multi-core CPUs.

In theory, this could give Suricata a performance advantage relative to the older Snort technology. (OISF, 2012)

Outside of the improved technology, Suricata shares many of the characteristics (both positive and negative) of other IDS options. Suricata uses rules to perform both behavior and signature-based analysis against known bad traffic. Users of Suricata are able to use the Snort rules language to build custom rules. In addition, there are other sources of pre-built rules including the Sourcefire VRT and Emerging Threats rule sets.

2.1.3 Host-Based Intrusion Prevention

According to Cisco, host-based intrusion prevention systems consist of “software installed on endpoints – desktops and servers – as opposed to network appliances”. (Cisco, 2004) While this is a good general definition, there is much more to it than that. Previously in this paper the three main types of IPS were discussed. As time goes on, more and more firms are moving to the host-based model. This trend was highlighted and brought mainstream when Microsoft released service pack 2 for Windows XP. This service pack shipped with the Windows XP “personal firewall” enabled by default. While this personal firewall is basic in terms of customizability and features, at its core it is a host-based system for combating intrusion.

According to research performed by Gartner, there should be a three-layer model that encompasses nine protection styles of HIPS.

	Allow Known Good	**Block Known Bad**	**Unknown**
Execution-Level	Application Control	Resource Shielding	Behavioral Containment

Application-Level	Application and System Hardening	Antivirus	Application Inspection
Network-Level	Host Firewall	Attack-Facing Network Inspection	Vulnerability-Facing Network Inspection

This model moves from left to right and once again illustrates that HIPS is not an independent solution, but a process that requires attention from the networking, security, and application support disciplines. In general, as you move from left to right, the risk of false positives increases as the risk of false negatives decreases. (MacDonald, 2006)

While generally very effective, host-based IPS do have some weaknesses. As discussed earlier host-based IPS do require a greater investment in terms of computing resources and power to perform the deep packet analysis. Also, HIPS with active response capability and/or active heuristic scanning require more CPU than the average IPS. As a result, from an infrastructure standpoint a firm needs to make sure that their host hardware is as up to date as possible. Lastly, because the HIPS is the last line of defense for a host, there can be a single point of failure. While a single host failure is rare due to the interaction between the IPS and the application binary interface (ABI), if the HIPS fails, the host will be compromised. The ABI works to define the application program language and the specific machine language for each processor and chipset. Because the ABI operates at the binary level, it is not subject to application level exploits. (Amarasinghe, 2005)

Another drawback to the HIPS is the initial implication cost. The financial cost of the technology was addressed earlier in this paper, but did not take into account the cost in human resources needs for an implementation of HIPS. As this is a host-based system, every host in the network will need to be protected. In a firm with a large, complicated network that can be a tall order. For example, in a network with 3,500 workstation and 600 servers there will be over 4,000 machines that need to be addressed. That can create resource challenges from a man-hours perspective. Further, in the beginning a firm can expect some rough spots while the analysts get the system properly tuned and configured for the environment. In the worst case, this could lead to down time and lost productivity.

Outside of the technical aspects of the system, there are a couple of larger issues to factor in. Decision makers need to take a hard look at the environment and the culture of their firm to ensure that a system like this is a good fit. While HIPS is an excellent tool in any information security team’s arsenal, it is not a “silver bullet”. Too many firms make poor decisions that lead to systems that are “not quite right” for their environment. After purchase and implementation, they wonder why the solution isn’t working the way “the vendor said it would”. After a while they give up on the system without realizing that the reason for failure was that the solution was not the right one for their environment and corporate culture.

2.2 Data Loss Prevention

Data loss prevention (DLP) products are a more recent evolution of the monitor and detect approach to defending networks. According to Gartner there are three main categories of DLP, enterprise, channel, and DLP-lite. (Ouellet, 2011) The difference between these three categories is largely based around scale and functionality; Enterprise DLP, not surprisingly, is the most fully featured (and most capital intensive) of the three categories. Typical enterprise DLP solutions provide a full-suite of context-aware monitoring and detection, including installed agents that have the ability to actively scan data repositories and monitor activity on the host. (Mogull, 2010) This functionality gives organizations the ability to block file transfer activity at the host, taking away the ability of most users to exfiltrate data using USB drives or other high-capacity portable storage media.

Channel DLP offerings are less fully featured and are mostly bundled as part as a different solution. These solutions are typically point solution specific; addressing a particular use case specific to the application in which they are embedded. An example of this would be channel DLP technology installed as part of an email server; the embedded component could scan outgoing mail for sensitive or other prohibited data. (Ouellet, 2011)

DLP-lite offerings are mostly watered-down versions of the first two categories. Intended for smaller or less mature organizations, these offerings are limited by capability and intended to address a point need or very specific use cases. (Ouellet, 2011)

Regardless of which category of DLP is used, there is common functionality to all three. First, a capability to perform analysis of network traffic; similar to IDS/IPS technology, DLP is designed to sniff network traffic and alert on pre-determined "bad traffic" . In this case, bad traffic should be specifically defined by policy and well understood by the organization. Most DLP solutions provide the ability to both alert on or block bad traffic. Again, actions taken by the DLP solution should be spelled out specifically in organizational policy.

The other core functionality to most DLP solutions is the ability to interact with the host in a way that prohibits users from taken prohibited actions; one example of this would be copying of unapproved files as mentioned above. (Mogull, 2010) This functionality is provided through the installation of an agent on the monitored host. The agent acts a shim between the user and the operating system, monitoring activity and blocking prohibited actions.

DLP solutions are subject to similar weaknesses as IDS/IPS technologies. From a network perspective, they can struggle with encrypted traffic. Many DLPs provide the ability to decrypt and analyze SSL or VPN traffic, but are not able to do much with data that's encrypted outside of that type of point-to-point encryption. Having the installed local agent helps address that weakness; in a DLP-lite or channel solution, however, there may not be a local agent installed.

2.2.1 OpenDLP

OpenDLP is an open-source project designed to provide the ability for users to identify sensitive data at rest. This functionality is provided through both agent-based and agentless methods. The agentless method uses authentication over SMB to enable Windows file system and share scanning. OpenDLP does not provide the ability to monitor network traffic or perform any host-based interception or blocking of user action.

Despite the fact that OpenDLP lacks some of the more proactive host-based features common to more robust DLP solutions, it's flexibility and ease of use makes it an excellent candidate for inclusion in this project.

2.3 Leveraging IDS/IPS/DLP technologies as a HIDS?

The concept for this project was simply to find a way to create a workable HIDS with basic attack and data loss detection functionality for little to no cost. In researching for this project, the author thought back to a comment once made by a SANS instructor. The instructor was talking about ways to defend a Windows domain controller. In his opinion, one approach was to create the server as a virtual machine and only turn it on for the minimum amount of time needed to replicate the active directory, schema, and other components. His rationale was that in many situations, the DC is the most important server in the organization. If an attacker were able to obtain physical or remote domain admin access, enabling a simple IPSEC rule of "drop any any" would cripple most organizations in under an hour. Without good offsite backups, the organization would have a long road to recovery.

Extending that logic, then, if virtualization would work for a Windows DC, it should be workable for an IDS. By leveraging Sourcefire's VRT ruleset along with some custom rules written to demonstrate Snort's ability to use PCRE to detect clear text data strings in monitored network traffic, one could build a virtual machine capable of performing detection of attack traffic and prohibited data moving around the network. In addition, by adding OpenDLP's ability to perform identification of prohibited data at rest an organization could deploy a series of low-cost, easily maintainable virtual appliances without having to invest time and resources in larger Enterprise-grade solutions.

Certainly, there are gaps in this approach; namely, this solution would not be able to detect encrypted traffic. This limitation could be mitigated through the use of SSL proxies or other technologies able to man-in-the-middle encrypted traffic before it leaves the network. If successful, the appliance created through this project would allow small to mid-size organizations an option to increase their ability to detect malicious traffic on their networks while supporting the ability to respond to compliance requirements at a lower cost.

3.0 – Building the Appliance

The author's first two attempts at building the virtualized system failed miserably. The first attempt was on a Windows 7 laptop running MS Virtual PC. Although there are references to people successful running non-Windows guest OS's in Virtual PC, the achieved functionality fell short of "production" quality performance.

The author's second attempt was on a Windows 7 laptop running VMware Workstation. For this attempt Sun Solaris 10 was installed as the guest OS. The operating system installed fine, but installing the packages needed for Snort or Suricata to run, there was difficulty resolving many of the dependencies needed to make everything compile properly. On multiple occasions, a binary would be compiled and installed, only to have to be recompiled later because of a missing dependency. Again, as Snort is open source, documentation is spotty and there is no official support for installs or implementation.

The final installation attempt was much more successful. For this attempt a Mac OS X Mountain Lion laptop running VMware Fusion 5 was used. For this install, the minimal install of CentOS 6.3 was used as the host OS. The minimal install was selected simply for the ability to only include packages required for this use case as they were needed.

Once the core OS was loaded the yum package management program was used to download and install some of the required package. Yum makes installation very efficient because it automatically resolves dependencies and installs everything that is needed for an installation. After installing the CentOS session on the Mac OS X laptop, the author noticed a small performance decrease. In order for this to be a viable solution in production, the host machine would have to have sufficient resources to handle that increased load.

The remaining packages were manually installed from source code. PCRE and Libpcap are both required for Snort to work properly. PCRE allows for Perl regular expression matching with the Snort rule base, while Libpcap allows Snort to place the NIC in promiscuous mode for packet sniffing. Snort was installed next, specifically compiled to interface with MySQL. On a separate server, the BASE application was installed to be used as the graphical user interface (GUI) for testing. Of course, having a separate MySQL database and BASE instance on each machine simply doesn't scale. Both Snort and OpenDLP rely on a MySQL database backend. Many commercial log aggregators or security event management systems support polling MySQL databases. This allows for easy integration with such systems for alerting, event correlation, and reporting. A ticketing system or other notification method could use be utilized for close to real-time notifications.

Once everything was installed, the individual components were tested to make sure the executables were in the proper path and that everything started properly. Once everything appeared to be working properly, testing of the implementation began.

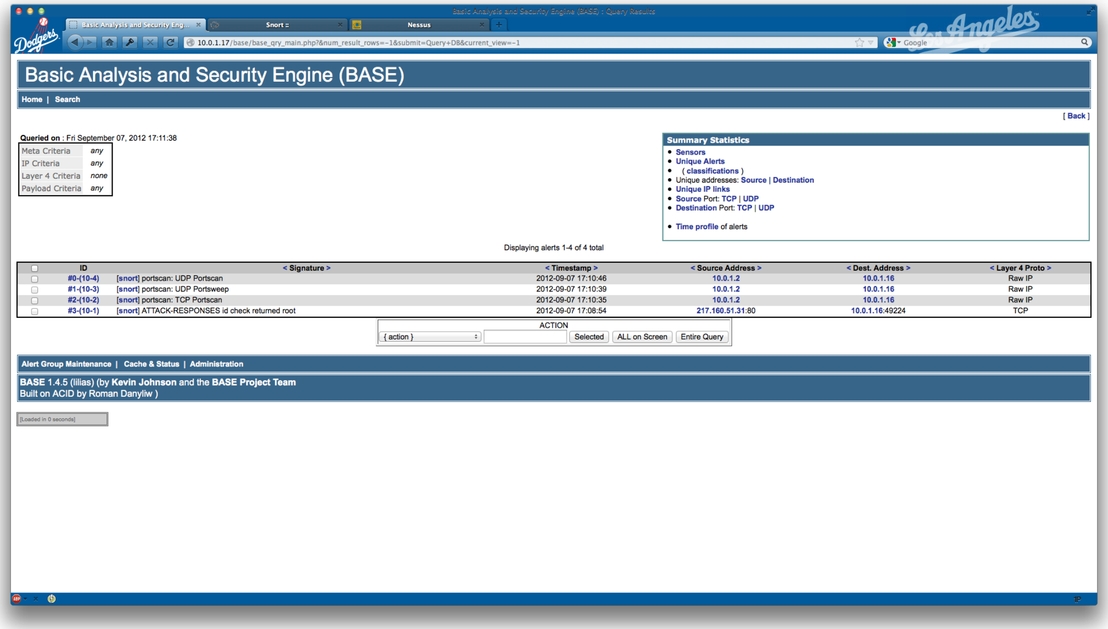
3.1 Testing the Snort VM Session

Prior to beginning testing of the appliance, there were two main concerns that needed validation. The first was performance; from a HIDS perspective, execution on server class hardware would not likely present an issue; most modern servers possess higher specs that what is required by the Snort application. Workstations, however, pose a slightly difference challenge. Working under the assumption that this appliance would likely be run on a system running a Windows operating system, the virtual appliance was limited to a single processor core with 512MB of RAM.

In most IDS/IPS deployments the devices are generally single purpose appliances built on a server class hardware platform or server class hardware running an IDS/IPS application suite. As discussed above, the desired use case in this instance is much different; this could lead to an unacceptably high number of dropped packets. If the appliance was unable to keep up with wire speed, it would have very little utility. The second concern relating specifically to the intended use case was control. In order for this approach to be effective, it would have to remain both isolated and unobtrusive. If the end user had the ability to interrupt the operation of the appliance, visibility into that node would be lost. To achieve this, the appliance was tested from within VMWare Player and run as a non-interactive, privileged account. Due to the limitations of this type of hardware and the specialized nature of the use case, performance testing was conducted to simulate the conditions in which this virtual appliance would be used.

For testing purposes, two tools were chosen. The first was Nessus 5.0 by Tenable Security. Nessus is a well-known vulnerability scanner that can be used to scan hosts on a network against commonly known vulnerabilities in various applications, operating systems, and network devices; the author thought this would be an effective test of the Snort rule base.

For the initial scan, the host (HOST A) running the virtual snort appliance was targeted from a remote workstation (HOST B). After initiating the Nessus scan, alerts immediately began to populate the MySQL database.



Basic Analysis and Security Engine (BASE) : Query Results
Basic Analysis and Security Eng...
Snort ::
Nessus
10.0.1.17/base/base_qry_main.php?&num_result_rows=-1&submit=Query+DB&current_view=-1
Google

Basic Analysis and Security Engine (BASE)

Queried on : Fri September 07, 2012 17:11:38

Meta Criteria	any
IP Criteria	any
Layer 4 Criteria	none
Payload Criteria	any

Summary Statistics

- Sensors
- Unique Alerts
- (classifications)
- Unique addresses: Source | Destination
- Unique IP links
- Source Port: TCP | UDP
- Destination Port: TCP | UDP
- Time profile of alerts

Displaying alerts 1-4 of 4 total

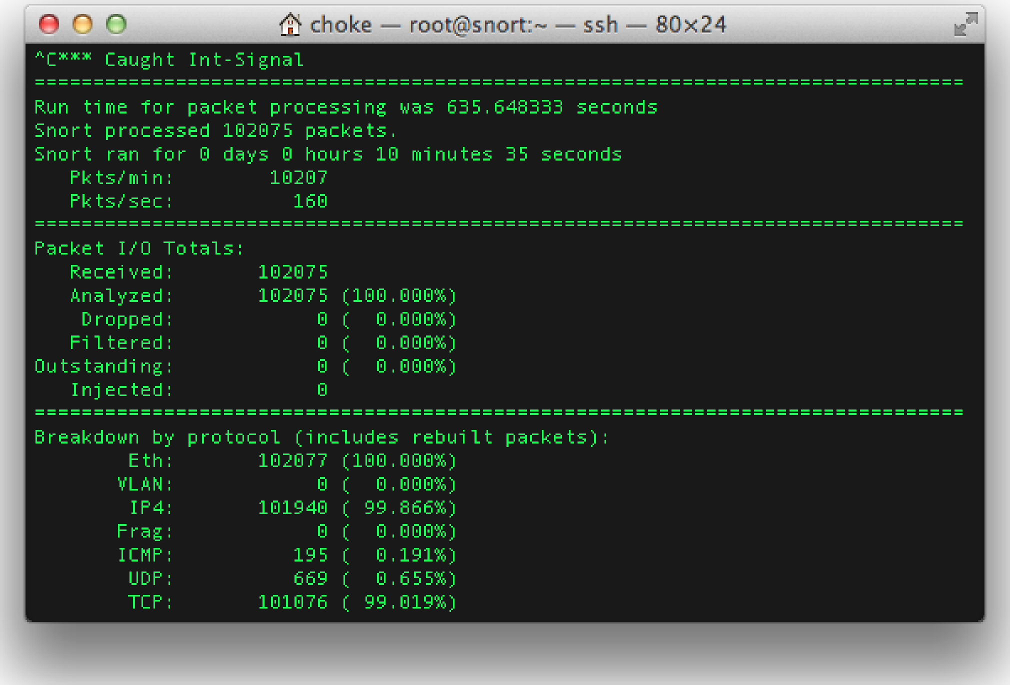
□	ID	< Signature >	< Timestamp >	< Source Address >	< Dest. Address >	< Layer 4 Proto >
□	#0-(10-4)	[snort] portscan: UDP Portscan	2012-09-07 17:10:46	10.0.1.2	10.0.1.16	Raw IP
□	#1-(10-3)	[snort] portscan: UDP Portsweep	2012-09-07 17:10:39	10.0.1.2	10.0.1.16	Raw IP
□	#2-(10-2)	[snort] portscan: TCP Portscan	2012-09-07 17:10:35	10.0.1.2	10.0.1.16	Raw IP
□	#3-(10-1)	[snort] ATTACK-RESPONSES id check returned root	2012-09-07 17:08:54	217.160.51.31:80	10.0.1.16:49224	TCP

BASE 1.4.5 (lilias) (by **Kevin Johnson** and the **BASE Project Team**
Built on ACID by Roman Danyliw)

[Loaded in 0 seconds]

Figure 1 – BASE showing first alerts from test scans

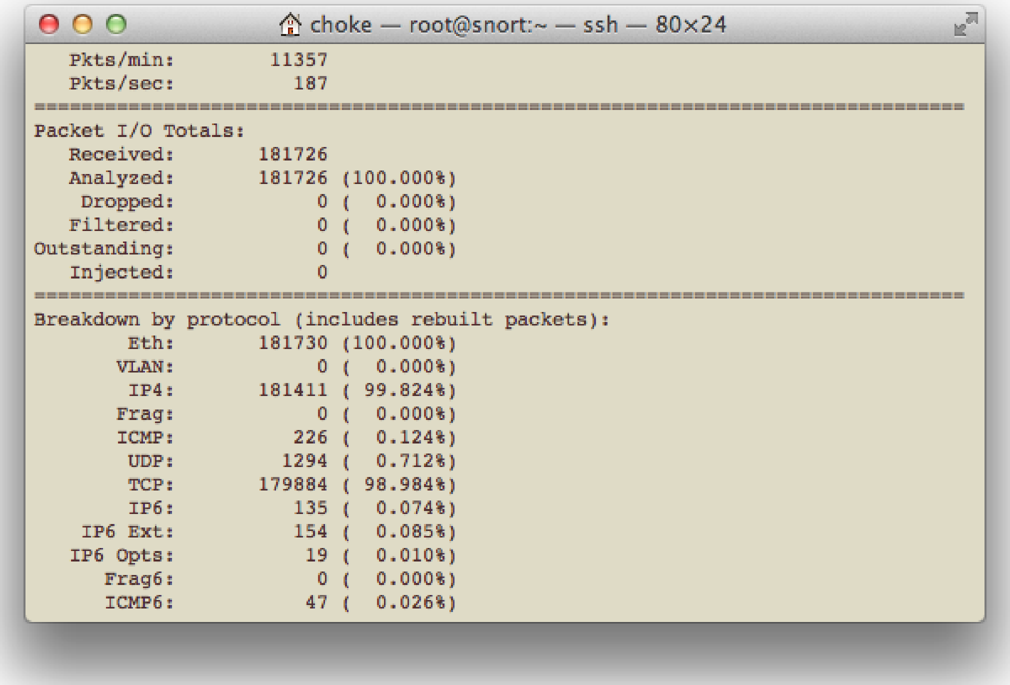
The majority of the alerts were for what would be considered common reconnaissance scans (nmap port scans, tcp half open scans, etc) and scans for well-known vulnerabilities. During this initial scan, the Snort engine saw 102,075 packets and had a 0% drop rate; that is, the Snort engine was able to process every packet that passed the interface.



```
^C*** Caught Int-Signal
===============================================================================
Run time for packet processing was 635.648333 seconds
Snort processed 102075 packets.
Snort ran for 0 days 0 hours 10 minutes 35 seconds
   Pkts/min:        10207
   Pkts/sec:          160
===============================================================================
Packet I/O Totals:
   Received:       102075
   Analyzed:       102075 (100.000%)
    Dropped:            0 (  0.000%)
   Filtered:            0 (  0.000%)
Outstanding:            0 (  0.000%)
   Injected:            0
===============================================================================
Breakdown by protocol (includes rebuilt packets):
        Eth:       102077 (100.000%)
       VLAN:            0 (  0.000%)
        IP4:       101940 ( 99.866%)
       Frag:            0 (  0.000%)
       ICMP:          195 (  0.191%)
        UDP:          669 (  0.655%)
        TCP:       101076 ( 99.019%)
```

Figure 2 - Snort successfully receiving and analyzing network traffic

For the next test, Nessus was again used. During the second test, concurrent scans from HOST A targeting HOST B and from HOST B targeting HOST A were run in an attempt to increase the level of traffic seen. Again, after starting the scans, alerts immediately began to populate the MySQL database. During the second test, the Snort engine saw 181,726 packets and again had a 0% drop rate. The alerts generated by these Nessus scans were what had been anticipated, and given the drop rate, the test could be declared a success.

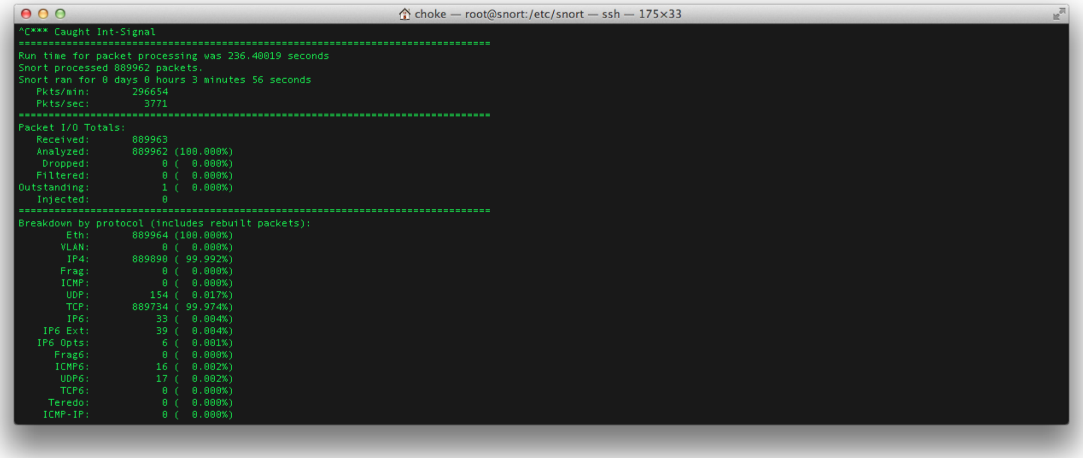


```
   Pkts/min:        11357
   Pkts/sec:          187
===============================================================================
Packet I/O Totals:
   Received:       181726
   Analyzed:       181726 (100.000%)
    Dropped:            0 (  0.000%)
   Filtered:            0 (  0.000%)
Outstanding:            0 (  0.000%)
   Injected:            0
===============================================================================
Breakdown by protocol (includes rebuilt packets):
        Eth:       181730 (100.000%)
       VLAN:            0 (  0.000%)
        IP4:       181411 ( 99.824%)
       Frag:            0 (  0.000%)
       ICMP:          226 (  0.124%)
        UDP:         1294 (  0.712%)
        TCP:       179884 ( 98.984%)
        IP6:          135 (  0.074%)
    IP6 Ext:          154 (  0.085%)
   IP6 Opts:           19 (  0.010%)
      Frag6:            0 (  0.000%)
      ICMP6:           47 (  0.026%)
```

Figure 3 – Snort maintaining a minimal drop rate against initial traffic

The second tool used for testing was Scapy. Scapy is a python platform that allows in-depth packet crafting and manipulation. Scapy allows the crafting and sending of packets containing a specific payload. To facilitate performance testing, the author created Snort rules designed to use the PCRE option. When used in Snort rules, the PCRE option uses regular expressions to perform pattern matching inside of the captured packet. This inspection requires more machine overhead to perform, and can lead to an increase in dropped packets. For this test, the Snort rules used were designed to match against numeric strings, modeling the rules after common patterns found in credit card numbers and US social security numbers. The rules used in testing are presented in Appendix A.

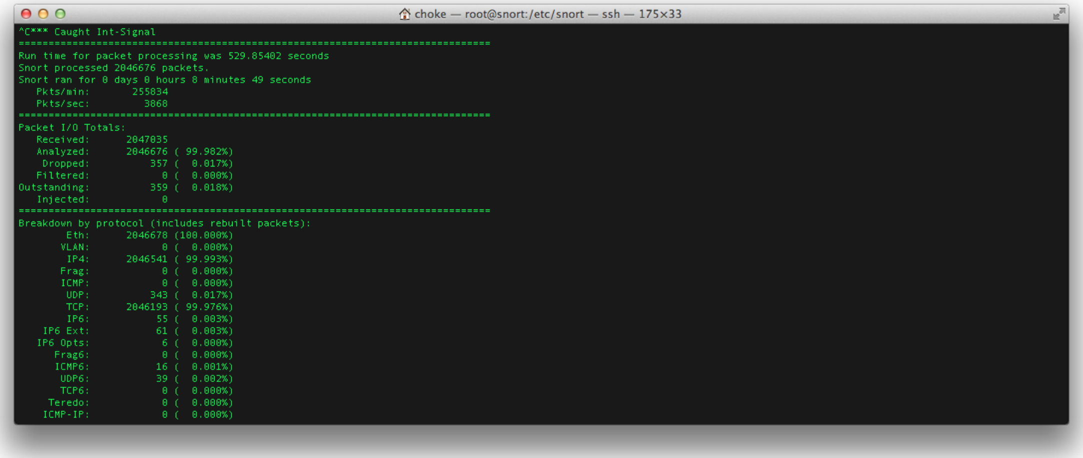
Once the rules were created, Scapy was configured to send a constant stream of packets while Snort ran for 3 minutes and 56 seconds. At the end of this run, Snort reported that of the 889,962 packets processed (a rate of 3,771 packets per second) it again had a 0% drop rate.



```
^C*** Caught Int-Signal
================================================================================
Run time for packet processing was 236.40019 seconds
Snort processed 889962 packets.
Snort ran for 0 days 0 hours 3 minutes 56 seconds
   Pkts/min:       296654
   Pkts/sec:         3771
================================================================================
Packet I/O Totals:
   Received:       889963
   Analyzed:       889962 (100.000%)
    Dropped:            0 (  0.000%)
   Filtered:            0 (  0.000%)
Outstanding:            1 (  0.000%)
   Injected:            0
================================================================================
Breakdown by protocol (includes rebuilt packets):
        Eth:       889964 (100.000%)
       VLAN:            0 (  0.000%)
        IP4:       889890 ( 99.992%)
       Frag:            0 (  0.000%)
       ICMP:            0 (  0.000%)
        UDP:          154 (  0.017%)
        TCP:       889734 ( 99.974%)
        IP6:           33 (  0.004%)
    IP6 Ext:           39 (  0.004%)
   IP6 Opts:            6 (  0.001%)
      Frag6:            0 (  0.000%)
      ICMP6:           16 (  0.002%)
       UDP6:           17 (  0.002%)
       TCP6:            0 (  0.000%)
     Teredo:            0 (  0.000%)
    ICMP-IP:            0 (  0.000%)
```

Figure 4 – Snort maintaining a minimal drop rate against increases levels of traffic

For the last test, the same rules were used, but Snort was allowed to run for a longer period of time. At the end of the last test, Snort reported that of the 2,074,035 packets processed (a rate of 3,868 packets per second) there was a .017% drop rate. It seems that virtualizing Snort does not disable any of the expected functionality. In addition, the performance demonstrated through this testing indicates that normal workstation behavior and usage should support the use of the virtual appliance with minimal packet loss.



```
^C*** Caught Int-Signal
================================================================================
Run time for packet processing was 529.85402 seconds
Snort processed 2046676 packets.
Snort ran for 0 days 0 hours 8 minutes 49 seconds
   Pkts/min:       255834
   Pkts/sec:         3868
================================================================================
Packet I/O Totals:
   Received:      2047035
   Analyzed:      2046676 ( 99.982%)
    Dropped:          357 (  0.017%)
   Filtered:            0 (  0.000%)
Outstanding:          359 (  0.018%)
   Injected:            0
================================================================================
Breakdown by protocol (includes rebuilt packets):
        Eth:      2046678 (100.000%)
       VLAN:            0 (  0.000%)
        IP4:      2046541 ( 99.993%)
       Frag:            0 (  0.000%)
       ICMP:            0 (  0.000%)
        UDP:          343 (  0.017%)
        TCP:      2046193 ( 99.976%)
        IP6:           55 (  0.003%)
    IP6 Ext:           61 (  0.003%)
   IP6 Opts:            6 (  0.000%)
      Frag6:            0 (  0.000%)
      ICMP6:           16 (  0.001%)
       UDP6:           39 (  0.002%)
       TCP6:            0 (  0.000%)
     Teredo:            0 (  0.000%)
    ICMP-IP:            0 (  0.000%)
```

Figure 5 – Snort maintaining a minimal drop rate against increases levels of traffic

3.2 Testing the Suricata VM Session

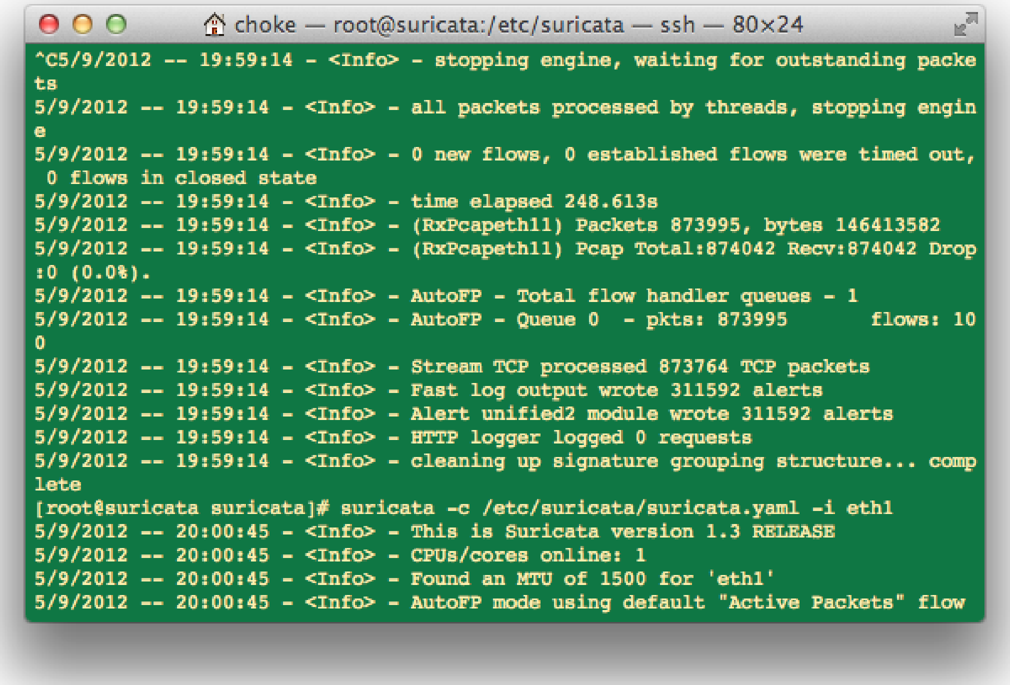
In testing the Suricata VM session, a similar methodology was followed as when testing the Snort VM. VMWare player was again used with a virtual machine configured with a single core and 512 MB of RAM to simulate the conditions in which this virtual appliance would be used.

For testing purposes, the same toolset and methodology was used. For the initial scan, the host (HOST A) running the virtual snort appliance was targeted from a remote workstation (HOST B). After initiating the Nessus scan, alerts immediately began to populate the MySQL database. The majority of the alerts were for what would be considered common scans (nmap port scans, tcp half open scans, etc) and well-known vulnerability detection; similar to what was seen in the Snort test, this was the expected traffic for the first test. During this initial scan, the Suricata engine saw 97,198 packets and had a 0% drop rate; that is, the Suricata engine was able to process every packet that passed the interface.

For the next test, Nessus was used again. For the second test, concurrent scans were executed from HOST A targeting HOST B and from HOST B targeting HOST A. Again, after starting the scans, alerts immediately began to populate the MySQL database. After this test was complete, the Suricata engine saw 226,553 packets and again had a 0% drop rate. The alerts generated by these Nessus scans were in line with expectations. Further, given the drop rate, the test could be declared a success.

The second tool used was again Scapy. Scapy is a python platform that allows in-depth packet crafting and manipulation. As in the first test, Scapy allowed the crafting of specific packets intended to cause the custom rules to fire. The same PCRE-based credit card and social security number rules used in the Snort testing were used for the Suricata performance testing.

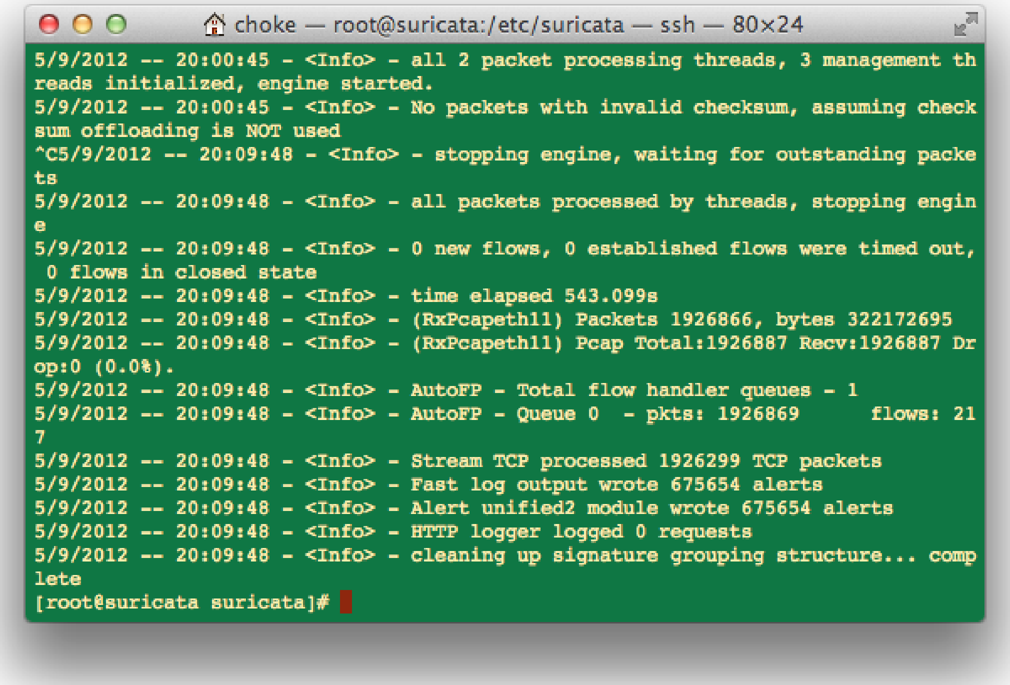
Once the rules were validated, Scapy was again used to create packets intended to trigger the new rules. Scapy was allowed to run 4 minutes and 00 seconds. At the end of this run, Suricata reported that of the 874,042 packets processed (a rate of 3,641 packets per second) it had a 0% drop rate.



```
^C5/9/2012 -- 19:59:14 - <Info> - stopping engine, waiting for outstanding packe
ts
5/9/2012 -- 19:59:14 - <Info> - all packets processed by threads, stopping engin
e
5/9/2012 -- 19:59:14 - <Info> - 0 new flows, 0 established flows were timed out,
 0 flows in closed state
5/9/2012 -- 19:59:14 - <Info> - time elapsed 248.613s
5/9/2012 -- 19:59:14 - <Info> - (RxPcapeth11) Packets 873995, bytes 146413582
5/9/2012 -- 19:59:14 - <Info> - (RxPcapeth11) Pcap Total:874042 Recv:874042 Drop
:0 (0.0%).
5/9/2012 -- 19:59:14 - <Info> - AutoFP - Total flow handler queues - 1
5/9/2012 -- 19:59:14 - <Info> - AutoFP - Queue 0  - pkts: 873995       flows: 10
0
5/9/2012 -- 19:59:14 - <Info> - Stream TCP processed 873764 TCP packets
5/9/2012 -- 19:59:14 - <Info> - Fast log output wrote 311592 alerts
5/9/2012 -- 19:59:14 - <Info> - Alert unified2 module wrote 311592 alerts
5/9/2012 -- 19:59:14 - <Info> - HTTP logger logged 0 requests
5/9/2012 -- 19:59:14 - <Info> - cleaning up signature grouping structure... comp
lete
[root@suricata suricata]# suricata -c /etc/suricata/suricata.yaml -i eth1
5/9/2012 -- 20:00:45 - <Info> - This is Suricata version 1.3 RELEASE
5/9/2012 -- 20:00:45 - <Info> - CPUs/cores online: 1
5/9/2012 -- 20:00:45 - <Info> - Found an MTU of 1500 for 'eth1'
5/9/2012 -- 20:00:45 - <Info> - AutoFP mode using default "Active Packets" flow
```

Figure 6 - Suricata successfully receiving and analyzing network traffic

For the last test, the same rules were used and Scapy was allowed to generate packets for 8 minutes. At the end of the last test, Suricata reported that of the 1,926,866 packets processed (a rate of 4,014 packets per second) there was a 0% drop rate. As with the Snort appliance, the performance demonstrated through this testing indicates that normal workstation behavior and usage should support the use of the Suricata virtual appliance with minimal packet loss.



```
5/9/2012 -- 20:00:45 - <Info> - all 2 packet processing threads, 3 management th
reads initialized, engine started.
5/9/2012 -- 20:00:45 - <Info> - No packets with invalid checksum, assuming check
sum offloading is NOT used
^C5/9/2012 -- 20:09:48 - <Info> - stopping engine, waiting for outstanding packe
ts
5/9/2012 -- 20:09:48 - <Info> - all packets processed by threads, stopping engin
e
5/9/2012 -- 20:09:48 - <Info> - 0 new flows, 0 established flows were timed out,
 0 flows in closed state
5/9/2012 -- 20:09:48 - <Info> - time elapsed 543.099s
5/9/2012 -- 20:09:48 - <Info> - (RxPcapeth11) Packets 1926866, bytes 322172695
5/9/2012 -- 20:09:48 - <Info> - (RxPcapeth11) Pcap Total:1926887 Recv:1926887 Dr
op:0 (0.0%).
5/9/2012 -- 20:09:48 - <Info> - AutoFP - Total flow handler queues - 1
5/9/2012 -- 20:09:48 - <Info> - AutoFP - Queue 0  - pkts: 1926869      flows: 21
7
5/9/2012 -- 20:09:48 - <Info> - Stream TCP processed 1926299 TCP packets
5/9/2012 -- 20:09:48 - <Info> - Fast log output wrote 675654 alerts
5/9/2012 -- 20:09:48 - <Info> - Alert unified2 module wrote 675654 alerts
5/9/2012 -- 20:09:48 - <Info> - HTTP logger logged 0 requests
5/9/2012 -- 20:09:48 - <Info> - cleaning up signature grouping structure... comp
lete
[root@suricata suricata]# 
```

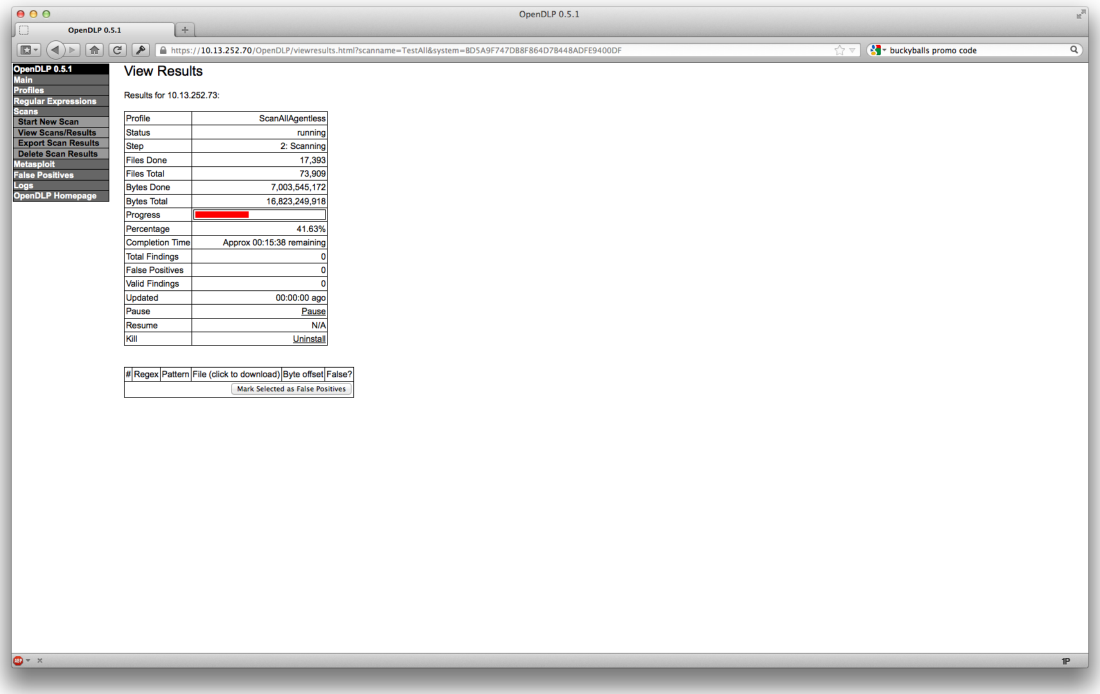
Figure 7 – Suricata maintaining a minimal drop rate against increases levels of traffic

Both the Snort and Suricata virtual machines performed well under load testing. The alerts generated by each solution were in line with initial expectations based on the testing tools used. Given the similarity of results, the author chose Snort as IDS/IPS tool of choice for this deployment in the short-term. Snort has a much stronger install base and community support. Over the long term, Suricata' s multi-threading ability could become an attractive feature to have, especially in I/O intensive applications.

3.3 – Testing OpenDLP

After validating both systems' operation, the author chose Snort as the system to use and OpenDLP' s agentless scanning functionality was tested. As discussed earlier, since OpenDLP is not running directly on the physical host, it does not provide the host integration that a Vontu or McAfee (Commercial DLP offerings) would include. One consequence here is that OpenDLP does not have the native ability to schedule scans. In order to consistently execute discovery scans of the host a scheduling facility, such as cron, would need to be used.

Figure 8 – OpenDLP Agentless scan of Windows 7 file system



https://10.13.252.70/OpenDLP/viewresults.html?scanname=TestAll&system=BD5A9F747DB8F864D7B448ADFE9400DF

View Results

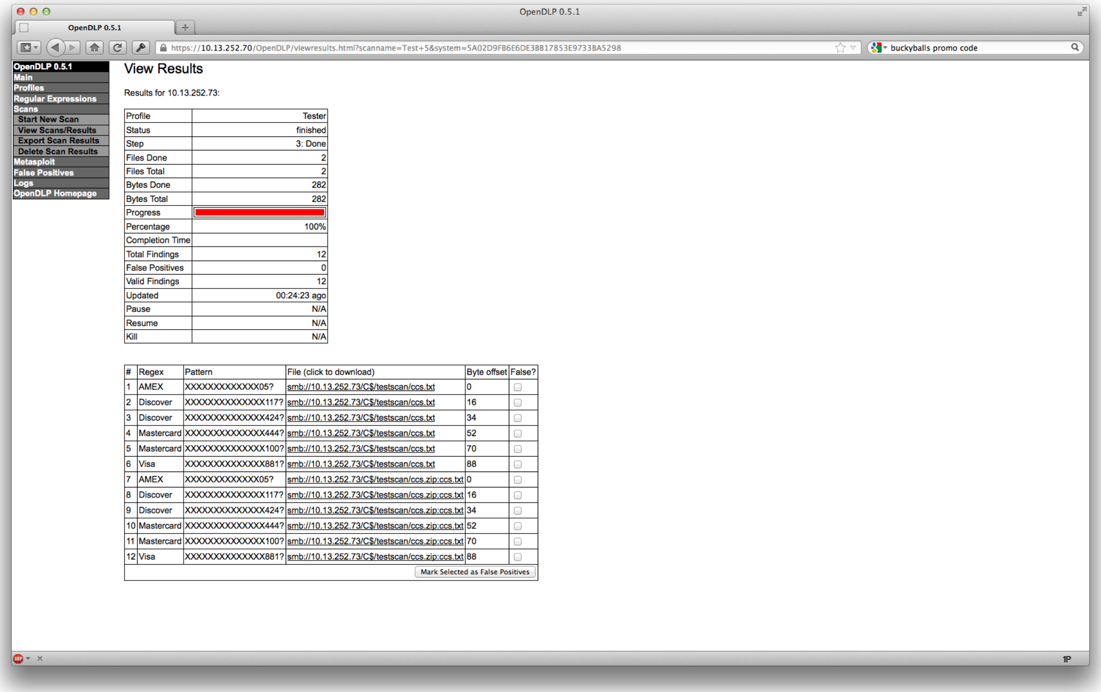
Results for 10.13.252.73:

Profile	ScanAllAgentless
Status	running
Step	2: Scanning
Files Done	17,393
Files Total	73,909
Bytes Done	7,003,545,172
Bytes Total	16,823,249,918
Progress	🟥
Percentage	41.63%
Completion Time	Approx 00:15:38 remaining
Total Findings	0
False Positives	0
Valid Findings	0
Updated	00:00:00 ago
Pause	Pause
Resume	N/A
Kill	Uninstall

#	Regex	Pattern	File (click to download)	Byte offset	False?
				Mark Selected as False Positives	

The Windows system tested was the host operating system, and was a fully patched Windows 7 workstation. In order to test full functionality, the author created text files containing credit card numbers, social security numbers, and dates of birth. Some of these files were compressed and some were encrypted. For the purpose of testing, the author used a local administrators account to allow OpenDLP to access the host system. According to the documentation, this credential is passed in clear text, so for other use cases an administrator would want to use a limited privilege user account. In this instance, however, the passing of credentials happens on the same physical machine making the transmission little more than inter-process communication.

As expected, OpenDLP was able to identify the contents of the uncompressed and compressed files, but not the files that had been encrypted.



View Results

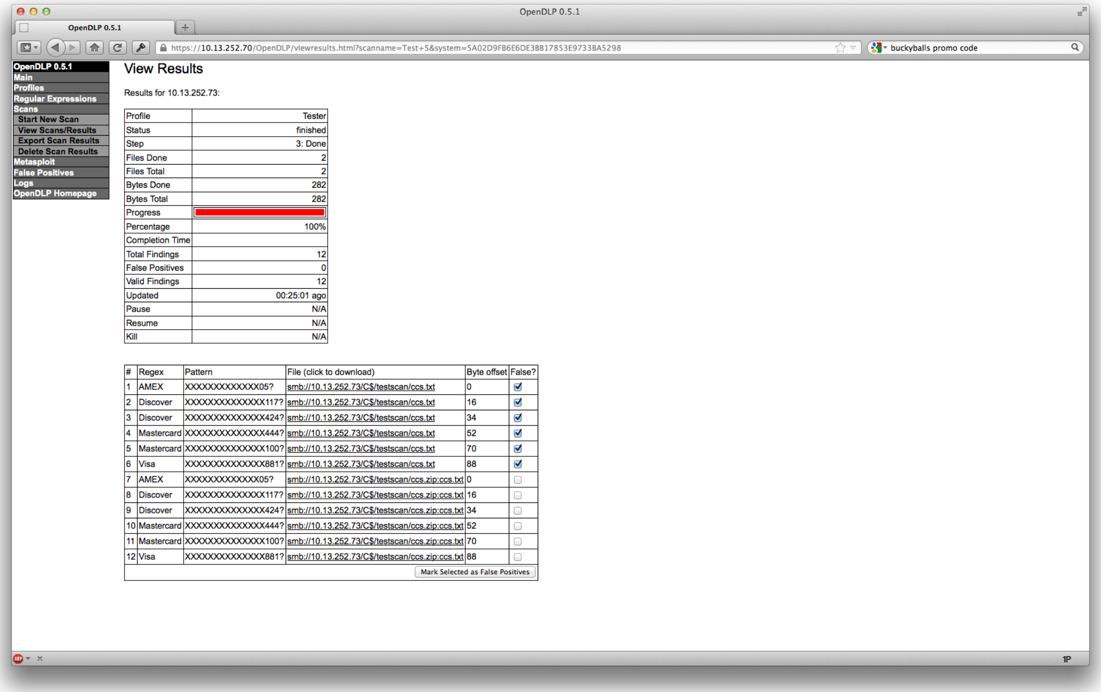
Results for 10.13.252.73:

Profile	Tester
Status	finished
Step	3: Done
Files Done	2
Files Total	2
Bytes Done	282
Bytes Total	282
Progress	🟥
Percentage	100%
Completion Time	
Total Findings	12
False Positives	0
Valid Findings	12
Updated	00:24:23 ago
Pause	N/A
Resume	N/A
Kill	N/A

#	Regex	Pattern	File (click to download)	Byte offset	False?
1	AMEX	XXXXXXXXXXXXX05?	smb://10.13.252.73/C$/testscan/ccs.txt	0	□
2	Discover	XXXXXXXXXXXXX117?	smb://10.13.252.73/C$/testscan/ccs.txt	16	□
3	Discover	XXXXXXXXXXXXX424?	smb://10.13.252.73/C$/testscan/ccs.txt	34	□
4	Mastercard	XXXXXXXXXXXXX444?	smb://10.13.252.73/C$/testscan/ccs.txt	52	□
5	Mastercard	XXXXXXXXXXXXX100?	smb://10.13.252.73/C$/testscan/ccs.txt	70	□
6	Visa	XXXXXXXXXXXXX881?	smb://10.13.252.73/C$/testscan/ccs.txt	88	□
7	AMEX	XXXXXXXXXXXXX05?	smb://10.13.252.73/C$/testscan/ccs.zip:ccs.txt	0	□
8	Discover	XXXXXXXXXXXXX117?	smb://10.13.252.73/C$/testscan/ccs.zip:ccs.txt	16	□
9	Discover	XXXXXXXXXXXXX424?	smb://10.13.252.73/C$/testscan/ccs.zip:ccs.txt	34	□
10	Mastercard	XXXXXXXXXXXXX444?	smb://10.13.252.73/C$/testscan/ccs.zip:ccs.txt	52	□
11	Mastercard	XXXXXXXXXXXXX100?	smb://10.13.252.73/C$/testscan/ccs.zip:ccs.txt	70	□
12	Visa	XXXXXXXXXXXXX881?	smb://10.13.252.73/C$/testscan/ccs.zip:ccs.txt	88	□

Figure 9 – OpenDLP Successful scan results

Additionally, OpenDLP identified a large number of false positives; files with strings that satisfied the regular expressions, but weren' t policy violations. Given the nature of OpenDLP’s regular expression searches, this was to be expected. For example, the most generic regular expression for SSNs used by OpenDLP is (\\D|^)[0-9]{9}(\\D|$). This regular expression will return any string of nine numbers. On initial scanning, this has the potential to generate a high number of false positives. As with many IPS products, OpenDLP needs to be “tuned” to identify and exclude false positives. Fortunately, OpenDLP has a web interface that allows false positives to be identified and managed; once a baseline is established, the tool seems to be very effective at locating privileged data.



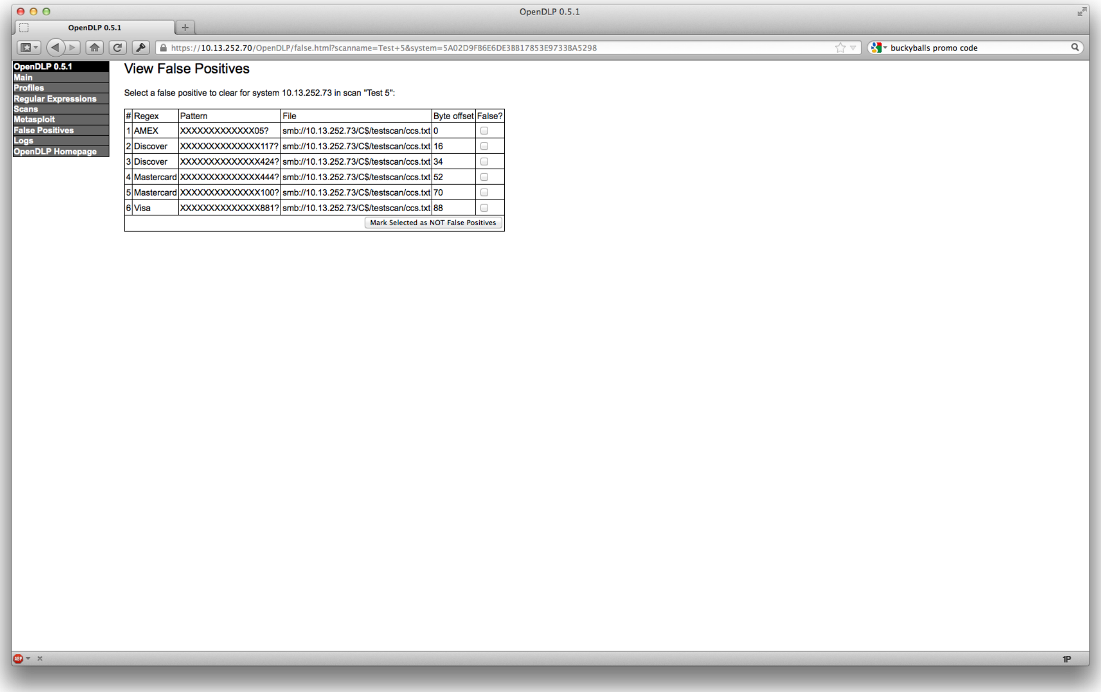
View Results

Results for 10.13.252.73:

Profile	Tester
Status	finished
Step	3: Done
Files Done	2
Files Total	2
Bytes Done	282
Bytes Total	282
Progress	🟥
Percentage	100%
Completion Time	
Total Findings	12
False Positives	0
Valid Findings	12
Updated	00:25:01 ago
Pause	N/A
Resume	N/A
Kill	N/A

#	Regex	Pattern	File (click to download)	Byte offset	False?
1	AMEX	XXXXXXXXXXXXX05?	smb://10.13.252.73/C$/testscan/ccs.txt	0	☑
2	Discover	XXXXXXXXXXXXXX117?	smb://10.13.252.73/C$/testscan/ccs.txt	16	☑
3	Discover	XXXXXXXXXXXXXX424?	smb://10.13.252.73/C$/testscan/ccs.txt	34	☑
4	Mastercard	XXXXXXXXXXXXXX444?	smb://10.13.252.73/C$/testscan/ccs.txt	52	☑
5	Mastercard	XXXXXXXXXXXXXX100?	smb://10.13.252.73/C$/testscan/ccs.txt	70	☑
6	Visa	XXXXXXXXXXXXXX881?	smb://10.13.252.73/C$/testscan/ccs.txt	88	☑
7	AMEX	XXXXXXXXXXXXX05?	smb://10.13.252.73/C$/testscan/ccs.zip:ccs.txt	0	□
8	Discover	XXXXXXXXXXXXXX117?	smb://10.13.252.73/C$/testscan/ccs.zip:ccs.txt	16	□
9	Discover	XXXXXXXXXXXXXX424?	smb://10.13.252.73/C$/testscan/ccs.zip:ccs.txt	34	□
10	Mastercard	XXXXXXXXXXXXXX444?	smb://10.13.252.73/C$/testscan/ccs.zip:ccs.txt	52	□
11	Mastercard	XXXXXXXXXXXXXX100?	smb://10.13.252.73/C$/testscan/ccs.zip:ccs.txt	70	□
12	Visa	XXXXXXXXXXXXXX881?	smb://10.13.252.73/C$/testscan/ccs.zip:ccs.txt	88	□

Figure 10 – OpenDLP Identification of false positives



View False Positives

Select a false positive to clear for system 10.13.252.73 in scan "Test 5":

#	Regex	Pattern	File	Byte offset	False?
1	AMEX	XXXXXXXXXXXX05?	smb://10.13.252.73/C$/testscan/ccs.txt	0	□
2	Discover	XXXXXXXXXXXX117?	smb://10.13.252.73/C$/testscan/ccs.txt	16	□
3	Discover	XXXXXXXXXXXX424?	smb://10.13.252.73/C$/testscan/ccs.txt	34	□
4	Mastercard	XXXXXXXXXXXX444?	smb://10.13.252.73/C$/testscan/ccs.txt	52	□
5	Mastercard	XXXXXXXXXXXX100?	smb://10.13.252.73/C$/testscan/ccs.txt	70	□
6	Visa	XXXXXXXXXXXX881?	smb://10.13.252.73/C$/testscan/ccs.txt	88	□

Figure 11 - OpenDLP list of identified false positives

From a performance perspective, the author was able to use the Windows host normally while the OpenDLP scan was running. Limiting the resources available to the virtual appliance running Snort and OpenDLP likely contributed to the host systems continued performance.

3.4 Overall Results

After testing the appliance, the author found that the open source components use here could be combined to create a monitoring tool that can be used to supplement other components of an organization' s network security monitoring program. From a performance perspective, the appliance ran well with one processor and 512 MB of RAM from inside of VMWare Player on a 2.8 GHz Intel i5 based Windows 7 desktop with 4 GB of RAM. As shown in the performance testing, the appliance should easily keep up with normal usage patterns.

In spite of the success of the tests, there are a couple of issues to be aware of. First, due to the nature of virtualization software there can be differences in the way the TCP/IP stack and networking are treated from host OS to host OS. For example, the author was involved in port scanning and fingerprinting a normal corporate environment. When scanned from a virtualized Linux running on a Windows host, the results indicated a large number of obscure South Korean digital cameras present on the network. Running the same test from a virtualized Linux running on Mac OS X resulting in proper identification of the target devices. Idiosyncrasies in how the TCP/IP stack is implemented by the host and virtualized can lead to unanticipated results. Adequate pre-deployment testing should ensure identification of these types of issues.

Another potential challenge is malware that is virtual machine aware. The popularity of virtualization among security researchers and professionals has led authors of malware to implement checks in their products that can detect when run in virtualization and adjust its performance accordingly. Often, this means the malware simply won’t run as to avoid reverse engineering and other types of testing. While the virtual appliance is simply deployed to listen to network traffic over a shared interface, this research did not attempt to install malware to the host for testing.

Overall, an appliance similar to what was tested here could be a solid addition to a mature network security monitoring program. While this solution does have some gaps, for organizations lacking resources to procure and deploy HIPS and DLP solutions it would provide added visibility into host activity that they currently lack.

4.0 Conclusion

After testing, the author found that this virtual appliance could be used to supplement an organization’s monitoring efforts as it relates to aspects of both IPS and DLP. The solution, while not a silver bullet, could be used at the host to augment existing monitoring technologies without a significant capital outlay. It should be obvious that there is still a place for traditional technical controls (firewalls, traditional IDS, network access control, etc.) even after implementing network security monitoring technologies. With multiple layers to carry the load, each device may have a smaller role but is still an important part of the layered security approach. As noted through this paper, this solution does leave some gaps but could be part of a well-considered defense in depth strategy. An organization that deployed this type of appliance on laptops and workstations would have additional visibility into host-based activity that is lacking in many organizations.

From an economic perspective, this approach becomes a good solution for organizations trying to protect key resources without spending a lot on the technology. After deployment, management of these appliances becomes easy due to the fact that if a change needs to be made or a patch needs to be applied, it can be made once to the virtual disk. The new disk can then be copied to each machine that needs it with little reconfiguration. Also, because this solution is created from mostly freeware to be implemented on hardware that is already owned and in production, the implementation costs are primarily in man-hours.

Regardless of the low cost of this solution, there are limitations to what it can detect. As mentioned earlier, encrypted traffic is one of its weaknesses. The inability to monitor application to host machine communication is another. These limitations help to illustrate what was discussed earlier, that no technology or solution is a silver bullet. Each tool or technology, this one included, should be a piece of a comprehensive detection and response strategy. Defense in depth has been and will continue to be the most effective way to protect the network and information assets of any organization.

5.0 References

Amarasinghe, Saman (2005, July 25). Host-Based IPS Guards Endpoints. *Network World*, *[22(29)]*, 35.

Caswell, Brian, & Hewlett, Jeremy (2007). Snort Users Manual 2.6.1. Sourcefire, Inc.

Cisco. (2004). *Understanding and deploying host-based intrusion prevention technology*. Session SEC-2031 Networkers 2004. Retrieved from http://www.cisco.com/networkers/nw04/presos/docs/SEC-2031.pdf

Desai, Neil (2003, February 27). Intrusion Prevention Systems: the Next Step in the Evolution of IDS. *SecurityFocus*, Retrieved 2006, 07, 14, from http://www.securityfocus.com/infocus/1670

Farshchi, Jamil (2003, November 5). Wireless Intrusion Detection Systems. *SecurityFocus*, Retrieved July, 28, 2006, from http://www.securityfocus.com/infocus/1742

Labbe, Keith, Rowe, Neil, & Fulp, J.D. (2006). A Methodology for Evaluation of Host-Based Intrusion Prevention Systems and Its Application. *2006 IEEE Information Assurance Workshop* . [378-379].

Larsen, Jason, & Haile, Jed (2002). Understanding IDS Active Response Mechanisms. *SecurityFocus*, Retrieved July 20, 2003, from http://www.securityfocus.com/infocus/1540.

MacDonald, Neil (2006). Understanding Strengths and Weaknesses of Host-Based Intrusion Prevention Styles. *Gartner. G00137366*

Mogull, R., Securosis, & SANS, (2010). Understanding and selecting a data loss prevention solution. Retrieved from https://securosis.com/Research/Publication/report-data-loss-prevention-whitepaper

Northcutt, S, & Novak, J (2002). *Network Intrusion Detection*. Pearson Education.

OISF. (2012). *What is suricata?*. Retrieved September 2, 2012 from https://redmine.openinfosecfoundation.org/projects/suricata/wiki/What_is_Suricata

Ouellet, E. (2011). Magic quadrant for content-aware data loss prevention. *Gartner*, doi: ID Number: G00213871Paulson, Linda Dailey (2002, November). Stopping Intruders

Outside the Gates. *Computer*, *[35(11)*, [20-22].

Pescatore, John (2005).Intrusion Prevention Process Consists of Seven Steps. *Gartner. G001130373*,

Tolly, Kevin (2006, February 20). Space Invaders: You and WIPS. *Network World*, *[23(7)]*, [18].

Appendix A – Snort rules used for performance testing

```
alert tcp any any <> any any (pcre:"/4\d{3}(\s|-)?\d{4}(\s|-)?\d{4}(\s|-)?\d{4}
/";msg:"Possible Visa Detected";sid:1000002;)

alert tcp any any <> any any (pcre:"/5\d{3}(\s|-)?\d{4}(\s|-)?\d{4}(\s|-)?\d{4}
/";msg:"Possible MC Detected";sid:1000003;)
```

```
alert tcp any any <> any any (pcre:"/6011(\s|-)?\d{4}(\s|-)?\d{4}(\s|-)?\d{4}/";
msg:"Possible Discover Detected";sid:1000004;)

alert tcp any any <> any any (pcre:"/3\d{3}(\s|-)?\d{6}(\s|-)?\d{5}/";msg:"Poss
ible AmEx Detected";sid:1000005;)

alert tcp any any <> any any (pcre:"/3\d{3}(\s|-)?\d{2}(\s|-)?\d{4}/";msg:"Poss
ible SSN Detected";sid:1000006;)

alert tcp any any <> any any (content:” Confidential” ; nocase;msg:” Possible
Confidential Data exfil Detected” ;sid:1000007;)

alert tcp any any <> any any (content:” Privileged” ; nocase;msg:” Possible Confidential
Data exfil Detected” ;sid:1000008;)

alert tcp any any <> any any (content:” Proprietary” ; nocase;msg:” Possible Confidential
Data exfil Detected” ;sid:1000009;)

alert tcp any any <> any any (content:” Client” ; nocase;msg:” Possible Confidential Data
exfil Detected” ;sid:1000010;)

alert tcp any any <> any any (content:” Visa” ; nocase;msg:” Possible Confidential Data
exfil Detected” ;sid:1000011;)

alert tcp any any <> any any (content:” Mastercard” ; nocase;msg:” Possible Confidential
Data exfil Detected” ;sid:1000012;)

alert tcp any any <> any any (content:” MC” ; nocase;msg:” Possible Confidential Data
exfil Detected” ;sid:1000013;)

alert tcp any any <> any any (content:” American Express” ; nocase;msg:” Possible
Confidential Data exfil Detected” ;sid:1000014;)

alert tcp any any <> any any (content:” Amex” ; nocase;msg:” Possible Confidential Data
exfil Detected” ;sid:1000015;)

alert tcp any any <> any any (content:” Discover” ; nocase;msg:” Possible Confidential
Data exfil Detected” ;sid:1000016;)
```