The Security Onion Cloud Client

Network Security Monitoring for the Cloud

GIAC (GCIA) Gold Certification

Author: Joshua Brower, Josh@DefensiveDepth.com
Advisor: Dennis Distler

Abstract

With "cloud" servers continuing to become ever more popular, along with typical off-site servers (VPS/Dedicated), Network Security Monitoring (NSM) practitioners struggle to gain insight into these devices, as they usually don't have the ability to tap the network traffic flowing to and from the servers—To solve this problem, I propose designing a cross platform (Windows, Linux) NSM client that would integrate with Security Onion, a NSM-centric Linux distribution. Essentially, the NSM client would copy traffic (near real time) to the Security Onion Sensor, which would then process the data as it would any other network tap. This would allow NSM practitioners the visibility they need into their off-site servers that are not in a setting where a typical NSM setup would suffice.

1. Introduction

1.1. The rise of Network Security Monitoring

Network Security Monitoring (NSM) is the "collection, analysis, and escalation of indications and warnings to detect and respond to intrusions." (Bejtlich, 2004, p. 3-4) NSM has a long and storied history, which Richard Bejiltch and Todd Heberlein recount in detail in *The Practice of Network Security Monitoring*. (Bejtlich, 2013) The key point to consider is NSM principles have been in use since early 1988—But it hasn't been until the past couple years that mainstream security vendors have started integrating NSM principles into their products, as they have continued to see the uselessness of the "security" that is created by the lack of contextual data when trying to analyze IDS alerts. At its core, NSM is based on the fact that when the analyst is running down a potential incident, they need as much context as possible to be able to determine the severity, if anything, of the event. This means, using NSM principles, the analyst will capture various types of information:

- Alert Data (NIDS, HIDS)
- Full Content Data
- Session Data
- Statistical Data
- Extracted Content
- Metatdata

In a typical NSM deployment, this data would be captured through a network tap or switch SPAN, which would allow a copy of all network traffic to be sent to the NSM platform, and analyzed.

1.2. The "rise" of Cloud Servers, and the problem

With the rise of cheap Virtual Private Servers (VPS) and now "Cloud Servers," the traditional capture method outlined previously does not work, as the analyst does not have any kind of access to the underlying network infrastructure. The VPS and cloud vendors up until this point have not made an effort to allow access to underlying logs, much less physical/virtual

networking infrastructure that would allow a SPAN or a tap. There has been some attempts to deal with this issue from some high profile vendors, for instance, Snort on Amazon EC2, though from the look of it, it is not been in widespread use. ("*Quick Start Guide for using Sourcefire Snort on Amazon EC2*," 2010) (Jon486, 2011)

To have a defensible network, administrators charged with security must have pervasive network awareness, which can be defined as "…the ability to collect the network-based information—from the viewpoint of any node on the network—required to make decisions." (Bejtlich, 2005, p. 27) Not having the ability to collect network-based data from a significant set of critical servers will certainly impinge on the administrator's ability to defend the assets within their purview.

As a NSM practitioner, how does one deal with this thorny issue? This paper proposes a Security Onion-integrated, multi-platform solution that would allow the analyst to gain the visibility he needs into the VPS and cloud servers that he is monitoring.

2. Basic Design Considerations

2.1. Target Audience

Because this type of NSM server-side client is a somewhat-new concept, the target scope for the first iteration, and this paper, was that of a proof of concept for small to medium size organizations that have a few cloud servers, which see light to medium usage primarily as web servers. A typical scenario would be a small or medium-sized business that hosts the majority of its services in-house, but does not have the facilities needed to host a 99.99% uptime public website. Consequently, they contract with Amazon AWS for a few cloud servers to take care of their public website needs.

2.2. Core Design Principles

There were six core design principles that needed to be considered when designing this system. These design principles were the guiding light for the design of the Cloud Client, to make sure the proposed solution would be stable, secure, and maintainable.

2.2.1. Integration with Security Onion

Security Onion is an open-source NSM platform that has been around since late 2008, when it was originally based on the Securix NSMNow scripts. (Securix NSMNow, 2010) It is now one of the industry's leading NSM platforms, (open source or commercial). With that in mind, the first core design principle was that the Cloud Client could not be a standalone, fully featured NSM platform. Instead, it needed to be an add-on to an existing platform. Security Onion was an easy choice, as it is open source, easy to work with, and the primary developers of Security Onion were very open to the idea of a Cloud Client.

2.2.2. No compromise of NSM Principles

This core requirement goes hand in hand with the previous one: the Cloud Client must not compromise on NSM principles. In other words, instead of just generating IDS alerts, it must be able to capture all relevant types of NSM data and present it for analysis.

2.2.3. Multi-Platform

Another core requirement was that the Cloud Client needed to be cross-platform—specifically, Windows and Linux. Based on data from Netcraft, Windows and IIS host around 20% of the world's websites, while Linux and Apache/NGINX host around 65%. (Netcraft, 2013) It is clear the need for NSM across platform boundaries is great, and therefore this became a core requirement.

2.2.4. Open Source

The fourth core requirement was that of the underlying components of the Cloud Client be open source. Because the Cloud Client would be a part of the open source Security Onion distribution, there was a need to make sure that it would fit well under the project. Also, the hope was that others will see the need for Cloud Client, and contribute to it—something much more difficult to do if it was proprietary.

2.2.5. Security

The fifth core requirement was that the design and implementation "Do The Right Thing™" when it came to the securability of the actual design and implementation. For instance, using bcrypt for hashing passwords, and not using MD5 or SHA-1.

2.2.6. Long-Term Sustainability

The final core requirement was that the design and implementation must be able to be somewhat easily sustainable long term. Said another way, the components used in the Cloud Client should have a track record of being maintained. This requirement was very key—there was no sense in going through all the trouble of building something like this when one of its components hasn't had bug patches for three years, and may not ever again.

2.3. Design Alternatives

A number of designs were considered and discarded based on their inability to satisfy the core requirements.

First off, it became obvious that in order to not compromise on collecting all the needed NSM data, the best way to do this would be to perform full packet capture on the target machine, transfer that data to the Security Onion Sensor, and then pass/replay/copy that data to an interface that Security Onion was listening on. This would allow Security Onion to capture the traffic and run the analysis on it, without any extra tweaking; Security Onion wouldn't know this is actually a host-side network tap, and not a normal network tap.

One of the biggest design concerns quickly became apparent—How to efficiently move the captured traffic securely over to the Security Onion Sensor? A number of options were considered and then rejected.

The first option was WinPcap's Remote Capture Protocol (RPCAP). (WinPcap, 2010) The latest documentation at the time of publishing this paper (4.1.2) describes it as follows: *"This is an[sic] highly experimental feature that allows to interact to a remote machine and capture packets that are being transmitted on the remote network. This requires a remote daemon (called*

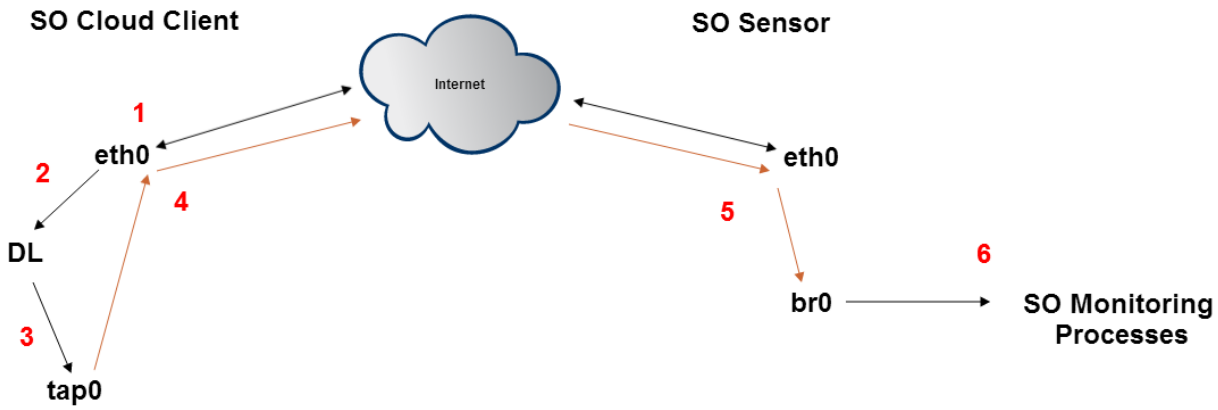
rpcapd) which performs the capture and sends data back and a local client that sends the appropriate commands and receives the captured data." (The WinPcap Team, 2010) There were also a couple other remote capture projects that had some potential. (ellzey, 2009) (Krishnan, 2002) All of the remote capture options that were considered had three major issues that led to them being disqualified: Firstly, they were alpha/experimental or did not seem to be under current development. Secondly, either there was no built-in encryption to secure the remotely captured traffic as it was sent back to the server, or the built in encryption was weak, which would have meant wrapping it in stunnel or some other encryption, which was highly undesirable. The final issue was that either the solutions were not multi-platform, or it was supposed to be, but in actually testing it out, by mixing a Windows and Linux remote capture client and server, the setup resulted in hacking code together, and a large amount of instability, which is not sustainable long term.

The next design option was to just have a scheduled job on the client copy over the captured data via SSH, and then replay or import the data into Security Onion. This method was rejected for two reasons: 1) Replaying the data using a tool such as Tcpreplay would overwrite the original timestamps on the network traffic, which is highly undesirable in any incident response situation, and 2) importing the pcaps into Security Onion would require a number of modifications to Security Onion itself, which was also highly undesirable.

The final option considered (and accepted) was to copy data over in near-real time via an OpenVPN bridge, to a Security Onion-monitored interface. This has a number of benefits: 1) highly secure copy mechanism (OpenVPN tunnel secured with certificates) 2) No need to store large amounts of pcaps on the local Cloud Client 3) No need to make any major modifications to Security Onion. The main concern with this option was, and still is, performance. Not only would we be capturing all network traffic, we would then be mirroring that back across the wire. The other major concern was that there was not a Windows packet capture and mirroring application like Daemonlogger, which is what would be used on Linux. After spending significant time searching and finding no viable options, a contract was posted online for a proof of concept. With generous donations from New Tribes Mission (ntm.org) and Defensive Depth Consulting, (DefensiveDepth.com) more than a proof of concept was developed—WinTAP has

been released under GPL 2, and can be found on GitHub (https://github.com/newtribesmission/wintap).

The Cloud Client Basic Design



The following numbers correspond to the diagram above, showing how the basic design works:

1) eth0 is the primary network interface for this server
2) Daemonlogger copies all traffic on eth0
3) Daemonlogger mirrors all of this traffic to the tap0 interface
4) The tap0 interface is part of the OpenVPN bridge, which is connected to the Security Onion Sensor
5) The traffic continues over the bridge into the Security Onion Sensor
6) The Security Onion processes on the SO Sensor monitor and analyzes the traffic that is coming across the br0 interface

3. In-Depth Design

As described previously, the chosen design consisted of using an OpenVPN bridge between the Security Onion Sensor and the Cloud Client.

3.1. Security Onion Sensor Design

The latest version of the Security Onion Sensor is officially supported on Ubuntu 12.04 LTS, and there are a number of quality guides on how to setup Security Onion, so this discussion will revolve around the Cloud Client-specific integration. (SO Documentation Team, 2013) (Bejtlich, 2013) At this time, the Cloud Client does not have an automated integration into Security Onion, but will in time. For now, a Debian package can be used to install and configure the sensor (it uses a set of shell scripts to automate the setup and configuration). The basic setup is as follows:

1) Prepare Ubuntu install like normal for a Security Onion install, but do not run SOSetup yet
2) Install OpenVPN Server and configure for use with certificates
3) Use bridge-utils and a pre-configured bridge-start script to create a persistent tap interface and add it to a bridge (eventually, this bridge will also include the tap from the other side of the VPN connection—See Appendix A)
4) Confirm that a firewall is not blocking traffic between the tap and bridge interfaces
5) Run SOSetup and configure as normal, making sure to select the bridge interface as a monitored interface
6) Startup OpenVPN in bridging mode

3.2. Cloud Client - Linux Design

The target OS for the Linux Cloud Client was Debian/Ubuntu, and there is a Debian package that automates the following:

1) Install Daemonlogger
2) Install and configure OpenVPN
3) User manually copies over previously generated OpenVPN certificates
4) Start OpenVPN in bridging mode to connect to Security Onion Sensor (this will create a tap interface)
5) Using ethtool, disable NIC offload features (These features will get in the way of full packet capture)
6) Start Daemonlogger with the options to mirror traffic from the desired interface to the newly created tap interface, along with a bpf setting that excludes all traffic from being captured unless

it is addressed to/from the Cloud Client, and it is not the OpenVPN port. This significantly cuts down on the amount of unneeded traffic from being captured and re-transmitted.

3.3. Cloud Client - Windows Design

The target version for the Windows Cloud Client was Server 2008 R2 & Server 2012. There is a batch script that automates the following:

1) Install WinTAP (requires the Visual C++ 2012 Redistributable x64)
2) Install and configure OpenVPN
3) User manually copies over previously generated OpenVPN certificates
4) Start OpenVPN in bridging mode to connect to Security Onion Sensor
5) Start WinTAP with the options to mirror traffic from the desired interface to the newly created tap interface. WinTAP does not have a bpf option yet, but it is currently under active development with an estimated release date of Q4 2013.

3.4. Bringing it all together

At this point, a couple simple tests can be run to confirm that all is working as it should. Browsing to http://testmyids.com on either of the Windows or Linux Cloud Clients should result in SID 2100498 *(GPL ATTACK_RESPONSE Id check returned root)* being generated from the installed NIDS. If this is not the case, on the Security Onion Sensor, use tcpdump to confirm that traffic generated on the Cloud Clients is being mirrored over to the Sensor. If it is, then the next steps would be to check the IDS and Security Onion configurations. If it is not, then go back and confirm that OpenVPN is configured correctly, Daemonlogger/WinTAP is running correctly, and that no firewalls are blocking any traffic.

3.5. Miscellaneous Design and Implementation Notes

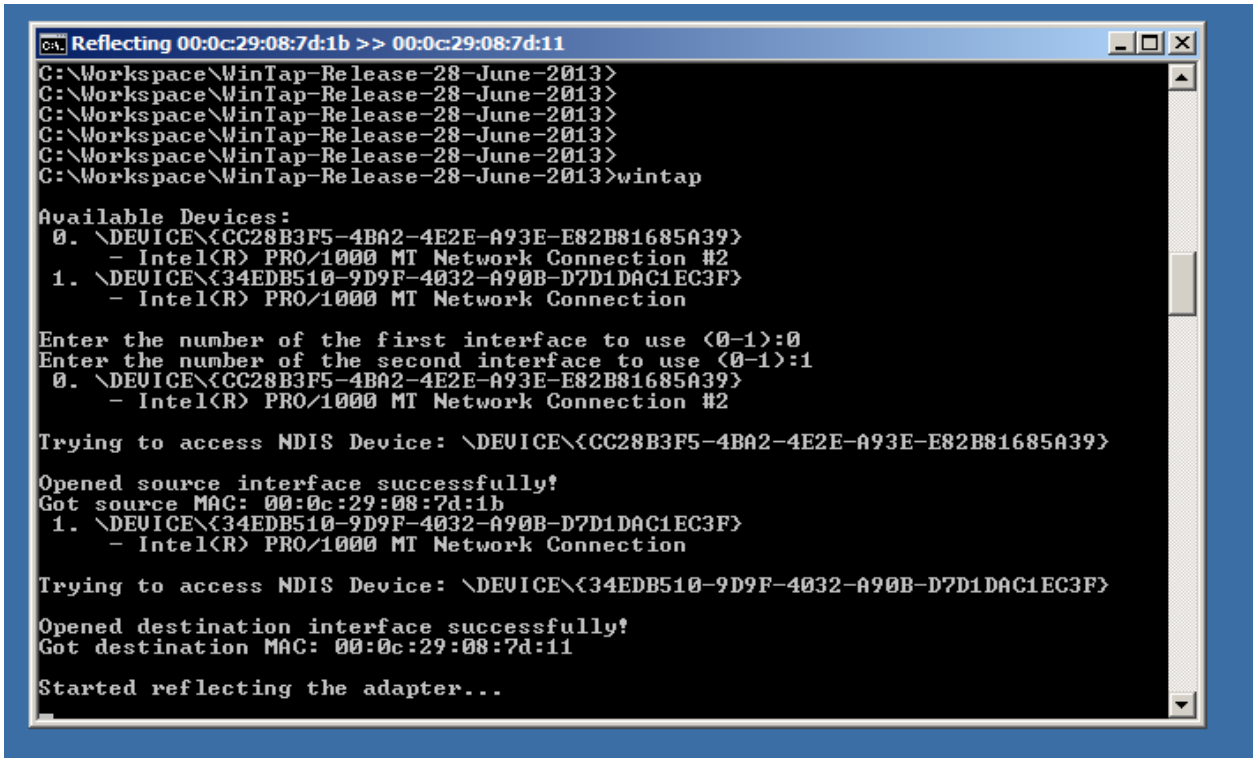
The following are some miscellaneous design and implementation notes to keep in mind.

3.5.1. WinTAP

WinTAP is the only component of this design that doesn't fit well with the core requirement of long term sustainability, as it is brand new, and quite experimental—Unfortunately, there was no way around this, as there was just no other applications for Microsoft Windows that emulated the soft tap features of Daemonlogger.

The first iteration of WinTAP was a proof of concept that was essentially a .NET wrapper around WinPcap. This worked well, but performance was not as well as could be hoped for. The next iteration took the proof of concept, and instead of using WinPcap, implemented WinTAP as a kernel mode driver, using NDIS 6.0. This allowed for much better performance and stability.

WinTAP in Use



3.5.2. OpenVPN Bridge Adapter Line Speed

A close observer will notice the OpenVPN bridged adapter for Windows state that it is running at 10Mbps. This has been well documented that this is not the actual line speed, but a "*Windows*

artifact from the days when ethernet interfaces were always hardware and never virtual." (Yonan, 2004)

3.5.3. Suricata Stream Alerts

In some circumstances where Suricata and the Cloud Client are being used together, there could be a very high rate of Suricata Stream alerts generated, which will bog down the Security Onion Sensor. The alerts will need to be tuned before much else is done. Refer to Richard Bejitlich's procedure on cleaning up and tuning these types of Suricata alerts. (Bejtlich, 2013)

3.5.4. Disabling NIC Offloading

In some cases, NIC offloading features need to be disabled to allow full packet capture to happen—See Doug Burk's write-up of this issue for more details. (Burks, 2011) Unfortunately, disabling NIC offload features may be more difficult than anticipated—Disabling these features on a stock Ubuntu Server 12.04.2 install was straightforward ("*ethtool --offload ethX rx off tx off"*), but disabling them on a stock Ubuntu Server 12.04.2 Rackspace Cloud Server install was very different. Running the same command resulted in the following error: *"Cannot set device rx csum settings: Operation not supported."* To get around this, there was a need to disable each offload feature individually ("*ethtool --offload ethX tso off", etc.),* then running "*ethtool --offload ethX rx off"* would work. It is not clear if this is a bug in the current Rackspace Ubuntu 12.04.2 image, or expected behavior.

3.5.5. IDS Home_Net Configuration

Whichever IDS is used, there is a need to edit the IDS Home_Net configuration, as the default variables are private networks, and unless the cloud servers are behind some sort of NAT that has been configured, no alerts will be generated, as the Cloud Client will be capturing traffic with public IPs.

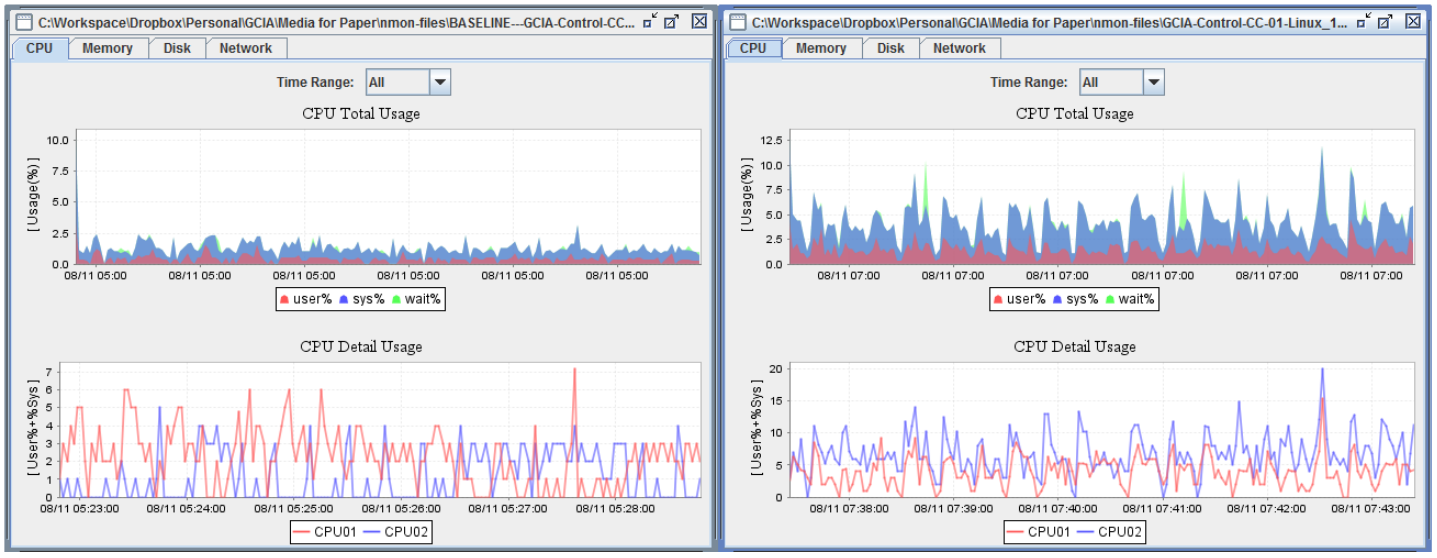
4. Performance & Stability Testing

4.1. Performance Metrics

To get a basic idea of what kind of performance hit this setup will make on a light to medium loaded webserver, nmon was used to generate performance snapshots over a five minute period on the Linux Cloud Client. At the time of this testing, WinTAP was not yet ready, so only Linux Cloud Client statistics were generated. Based off of Rackspace's Cloud Server template of 2GB RAM / 2 CPU, a local VM was created that mirrored that setup, and a vanilla LAMP stack with WordPress was setup. Constant loading of 10Mb/s was applied to the server. The graph on the left was a baseline five minute snapshot and the graph on the right is the baseline plus when Cloud Client is running (OpenVPN & Daemonlogger). Please note that memory is not graphed as there was not a measurable impact at this level.

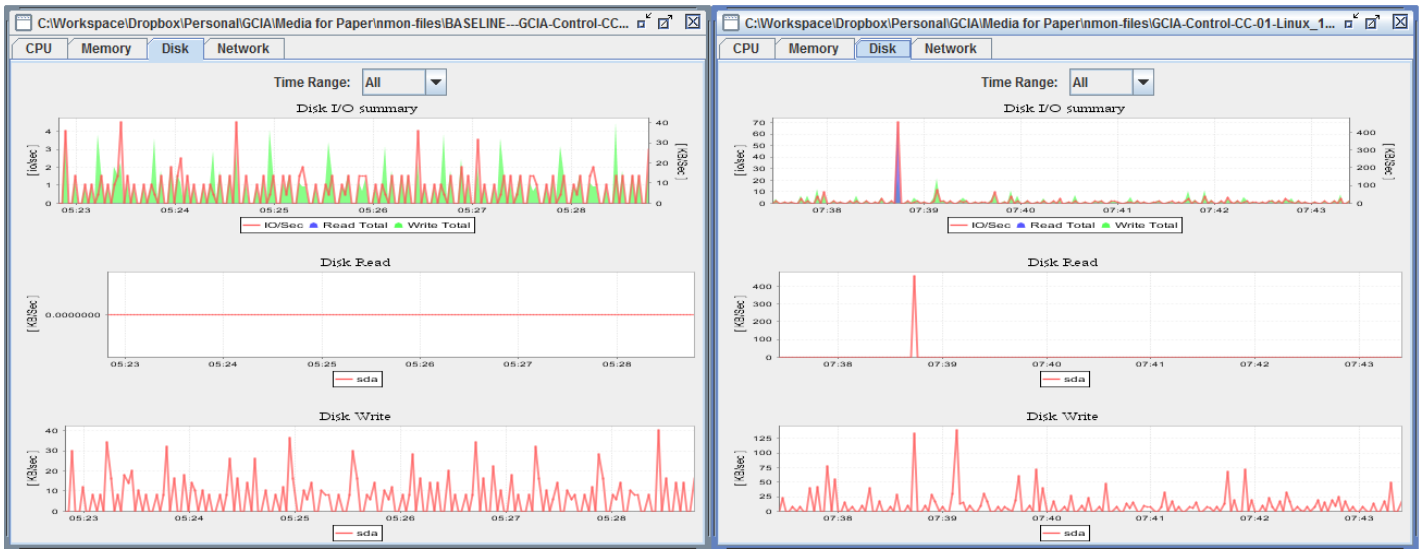
The raw nmon files are available for download and further analysis here:
http://DefensiveDepth.com/GCIA.zip

Cloud Client - CPU Usage



Baseline **Baseline + Cloud Client**

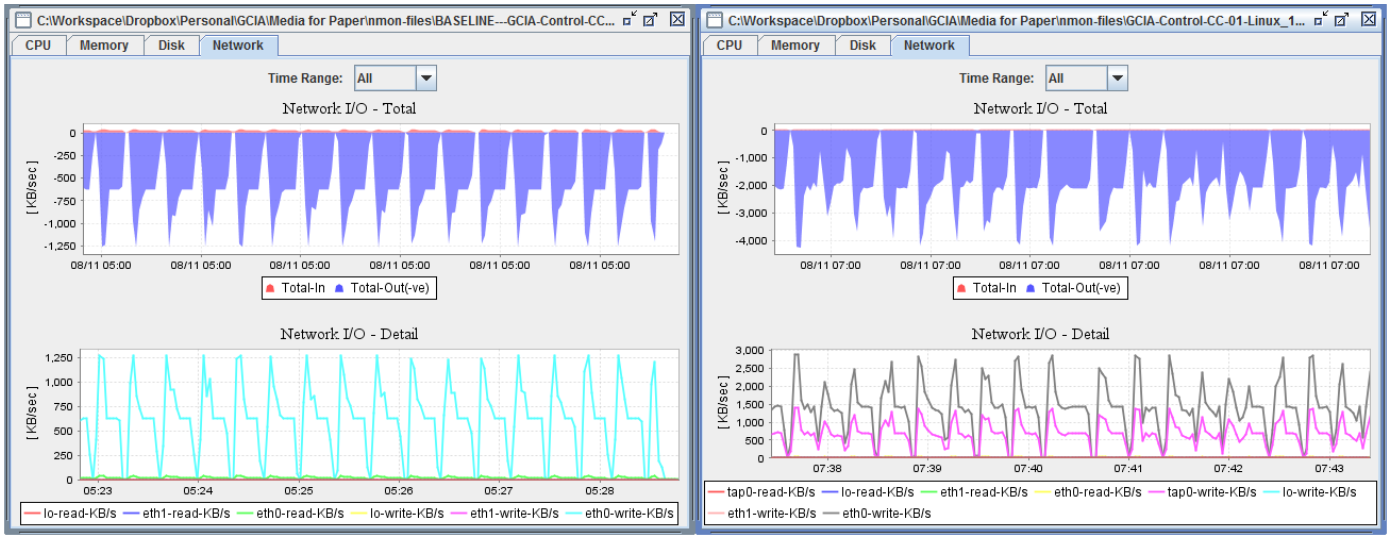
Client Client - Disk IO Usage



Baseline

Baseline + Cloud Client

Cloud Client - Network Usage



Baseline

Baseline + Cloud Client

As demonstrated, the Cloud Client is mirroring all traffic from eth0, (which is fulfilling http requests to the tune of 10Mb/s) and the difference between the baseline and the Cloud Client in network usage is to be as expected, nearly double 10Mb/s. By default, OpenVPN configures the

tunnel to use LZO compression, which with some types of data (http/text/JavaScript) will offer some good compression, and on other workloads (images/binaries) could be a detriment. (janjust, 2013)

Before running the Cloud Client on any kind of production load, there would need to be more testing on various configurations—this section was just to give a general idea of the performance hit for using the Cloud Client on a typical small webserver.

4.2. Stability

Because of the methodology used in the Cloud Client setup, there also existed a need to verify the overall architecture is stable, as well as validate the quality of NSM data collected was comparable to a traditional Security Onion setup.

With this in mind, three setups were constructed:

1) Traditional Security Onion Setup
This simulated a normal local Security Onion Setup whereby traffic was collected using a SPAN port on a switch. This was the control/baseline setup.

2) Local Cloud Client Setup
This simulated a Cloud Client setup via local (non-cloud) hardware, which was a VMware ESXi server with corresponding virtual machines.

3) Rackspace Cloud Client Setup
This simulated a true Cloud Client deployment, using Rackspace's Cloud Servers.

Each of the above setups had the following Servers:

1x Security Onion Standalone Sensor + Server
    Ubuntu Server 12.04.2 - 8GB RAM, 4 CPU, 320GB Disk, 100Mbps Network

2x LAMP Servers
    Ubuntu Server 12.04.2 - 2GB RAM, 2 CPU, 80GB Disk, 100Mbps Network
    Running Wordpress 3.5.1 + OWASP Mutillidae II

(WinTAP was not ready for use at the time of these tests.)

To simulate real-world use, 4.8 Mb/s of load was generated against the two LAMP servers, which meant close to 10Mb/s was incoming to the Security Onion Sensor.

Next, two types of tests were conducted:

1) Using OWASP Mutillidae II, SQLi and XSS vulnerabilities were exploited six times (three each against each LAMP server) over a five minute period.

2) Using NetSparker, a scan for Ruby on Rails and Remote/Local File Inclusion vulnerabilities was conducted six times (three each against each LAMP server) over a five minute period.

The resulting IDS alerts & NSM data was then analyzed. The IDS alerts generated can be seen in Appendix B. Analyzing the test data, it became clear all three setups generated the same quality of NSM data; there were no major concerns uncovered in either of the Cloud Client setups.

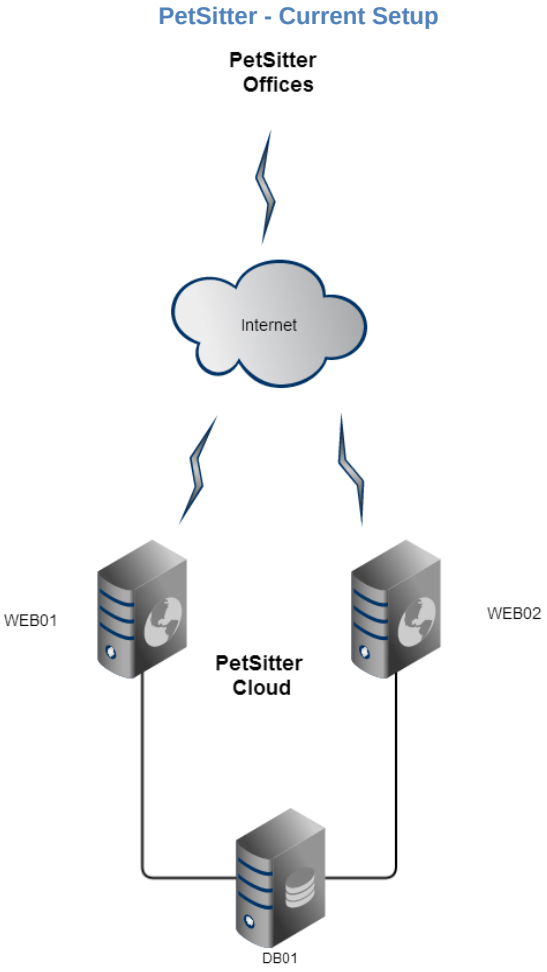
5. Security Onion Cloud Client in the real world

So how would Cloud Clients look in the "Real World?" What are some real world considerations to keep in mind?

5.1. Real World Scenario - PetSitter

Let us consider the fictional non-profit organization PetSitter. PetSitter exists to connect local pet owners with each other to trade off pet sitting while they are out of town. PetSitter runs a small server room in their offices, but it hosts only their local services—it was deemed unacceptable to host their mission-critical web application, which needs more stability than what their local server room can offer.

PetSitter has contracted with Rackspace to provide cloud servers for their web application, which is hosted on two load balanced Linux webservers, (WEB01/WEB02) connected to a backend database server. (DB01)



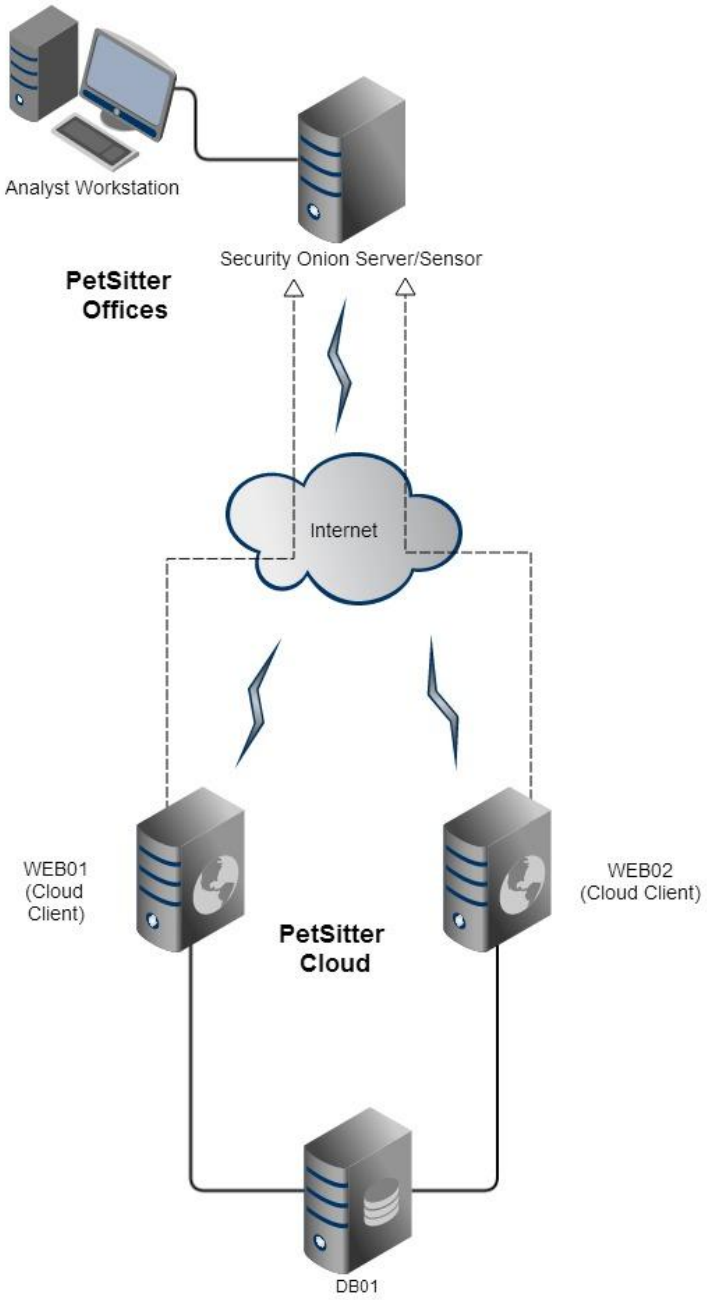
Because of the sensitivity of the pet and owner data PetSitter handles, they wish to gain more visibility into their office environment as well as the environment that hosts their PetSitter web application. Therefore, they have opted to implement Security Onion both at their local server room, and on their cloud servers, using Cloud Client as needed. Because the backend database server is sufficiently segmented, they have decided to include only WEB01 and WEB02 in the Security Onion scope for their cloud servers.

PetSitter has three basic architecture choices: Cloud Client/Local Server, Standalone Cloud, and Cloud Sensor/Local Server

5.2. Option one - Cloud Client, Local Server

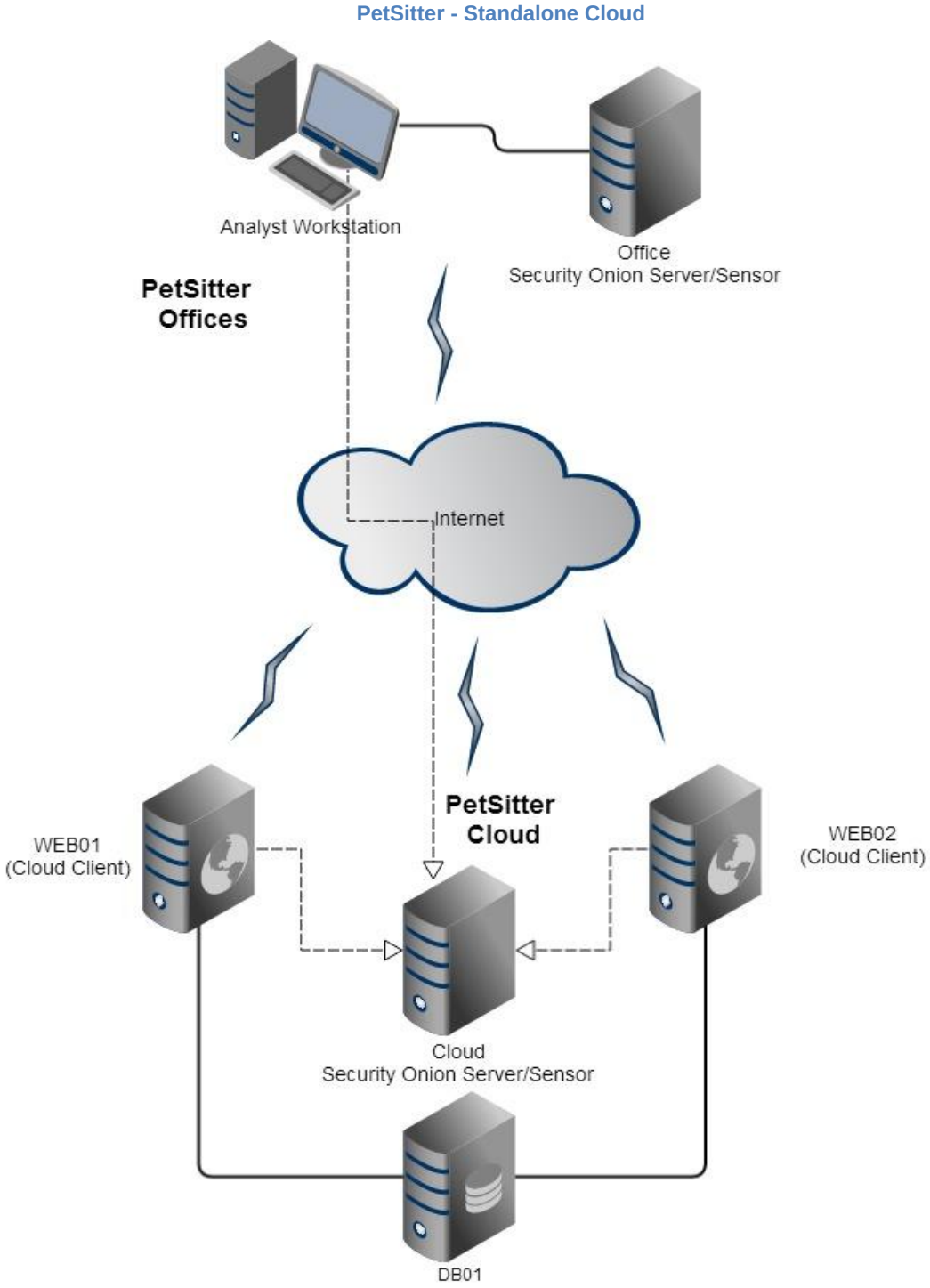
This option would require PetSitter to place a Cloud Client on WEB01 and WEB02 and have them connect to a Security Onion Sensor located at the PetSitter office server room. The primary reasoning for this is so that they don't have to run another Cloud server for the Security Onion Sensor.

PetSitter - Cloud Client, Local Server



5.3. Option two - Standalone Cloud

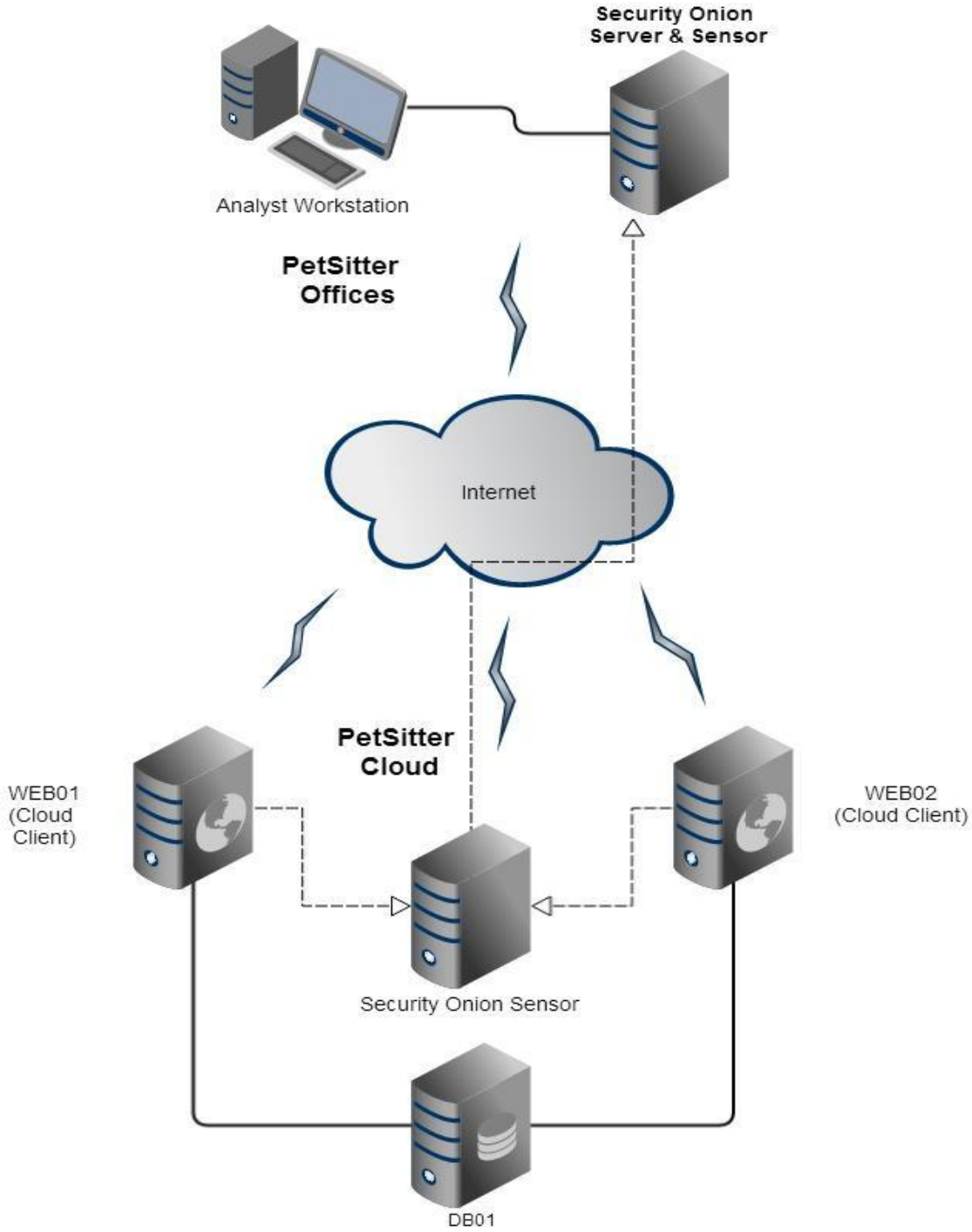
This option would require PetSitter to place a Cloud Client on WEB01 and WEB02 and have them connect via the private cloud network to a Security Onion Standalone Server (which includes the sensor) that is hosted within their Rackspace Cloud account. The NSM analyst would monitor and maintain two distinct instances of Security Onion—The office instance, and the cloud instance.



5.4. Option three - Cloud Sensor, Local Server

This option would require PetSitter to place a Cloud Client on WEB01 and WEB02 and have them connect to a Security Onion Sensor that is hosted within their Rackspace Cloud account. That sensor would then connect back to the Security Onion Server hosted at the PetSitter Offices.

PetSitter - Cloud Sensor, Local Server



5.5. Which Option?

Each architecture option has its pros and cons:

Option one would realize cost savings from not having to run a robust cloud server for a dedicated Security Onion Server/Sensor, but there would significant bandwidth costs, both outbound from the cloud servers from mirroring the traffic, and inbound into the PetSitter offices. Numbers would have to be crunched to see if it would be worthwhile to pay for this bandwidth, versus paying for a robust cloud server and keeping the bandwidth cost to a minimum. Another consideration is streaming that much data across the open Internet to the PetSitter offices will affect NSM data quality, as there will certainly be data loss in the transfer.

Option two would require the purchase of a robust cloud server to run the standalone Security Onion server, but bandwidth savings would be large, especially since the bandwidth that is being used to mirror WEB01 and WEB02 would go across the private cloud network, which is free (as long as the two instances are in the same datacenter and/or region). The primary drawback to this option is that since PetSitter is also running a Security Onion instance at its local office, the NSM analyst will have to maintain and monitor two different instances of Security Onion, duplicating her effort, and creating inefficiencies.

The third option is perhaps the best mix of both option one and option two—It allows for a less robust cloud server to act as a Security Onion Sensor, while keeping the bandwidth costs to a minimum by receiving the mirrored traffic over the private cloud network, and only sending small amounts of data outbound to the PetSitter office Security Onion server. It also allows the NSM analyst to have to maintain and monitor only one instance of Security Onion for both the local offices and their cloud servers.

6. Conclusion

The need for remote packet capture for security analysis has been foreseen by a number of people, particularly Richard Bejtlich, who wrote the following in *The Tao of Network Security Monitoring*, in 2004:

> In November 2001, and March 2001, Carter Bullard, (author of Argus) published Informational Internet Drafts on "Remote Packet Capture." Bullard claimed, "Packet capture is a fundamental mechanism in Internet network management and is used in support of a wide range of network operational functions, such as fault detection, protocol functional assurance, performance analysis, and security assessment." His proposal exceeded capabilities offered via RMON (RFC 2819, or Internet Standard 59) and SMON (RFC 2613), recognizing that switched networks are ubiquitous and sometimes full content data is the only evidence suitable for illuminating difficult network security and management problems… In the future I expect to see increased support in the open source world for collecting traffic remotely and seamlessly presenting it to a local workstation. (p. 652)

Cloud computing, in some form or fashion, is here to stay. Until the cloud computing vendors give organizations the needed visibility into the cloud environment that their services run in, there will be a need to get this data another way. This paper proposed a proof of concept solution to this issue—the Security Onion Cloud Client. For use in SMB situations where an organization has a few cloud servers that are in need of visibility, the Cloud Client will suffice. Design decisions, performance metrics, stability tests, and real-world architecture have all been laid out in this paper to help prompt further discussions and development on this particular topic.

7. References

Bejtlich, R. (2004). *The tao of network security monitoring: Beyond intrusion detection*. (1st ed. ed.). Boston: Addison-Wesley Professional.

Bejtlich, R. (2005). *Extrusion detection: Security monitoring for internal intrusions*. Upper Saddle River, NJ: Pearson Education, Inc.

Bejtlich, R. (2013). *The practice of network security monitoring*. (1st ed. ed.). San Francisco: No Starch Press, Inc.

Bejtlich, R. (2013, February 24). *Recovering from Suricata Gone Wild.* Retrieved from http://taosecurity.blogspot.com/2013/02/recovering-from-suricata-gone-wild.html

Burks, D. (2011, October 19). *When is full packet capture not full packet capture?.* Retrieved from http://securityonion.blogspot.com/2011/10/when-is-full-packet-capture-not-full.html

ellzey (2009, October). Rpcap - a remote pcap thingy [Software]. Available from https://github.com/ellzey/rpcap

janjust (2013, April 23). Re: OpenVPN compression for traffic reduction via satellite Message posted to https://forums.openvpn.net/topic12728.html

Krishnan, S. (2002, October). The RPCAP Remote Packet Capture System (Version .23) [Software]. Available from http://rpcap.sourceforge.net/

Netcraft. (2013). July 2013 Web Server Survey. Retrieved from http://news.netcraft.com/archives/2013/07/02/july-2013-web-server-survey.html

SO Documentation Team (Last updated, 2013). Installation. Retrieved from https://code.google.com/p/security-onion/wiki/Installation

Winpcap team (2010) *Remote Capture.* Retrieved from http://www.winpcap.org/docs/docs_412/html/group__remote.html

Yonan, J (2004, February 4). Re: [Openvpn-users] Win32 Tun/Tap adapter limited to 10Mbps???. Message posted to http://openvpn.net/archive/openvpn-users/2004-02/msg00041.html

(2010). Securix NSMNow (Version 1.6.2) [Software]. Available from http://www.securixlive.com/nsmnow/download.php

(2010, February 16). *Quick Start Guide for using Sourcefire Snort on Amazon EC2.* Retrieved from http://www.snort.org/assets/144/Snort_EC2_QuickStart.pdf

Jon486 (2011, April 27). Snort, IDS installation on Amazon Instance. Message posted to https://forums.aws.amazon.com/thread.jspa?threadID=66190

(2013). WinPcap (Version 4.1.2) [Software]. Available from http://www.winpcap.org/install/default.htm

Appendix A.

The customized bridge script, from OpenVPN's documentation website:

```
#!/bin/bash

###################################
# Set up Ethernet bridge on Linux
# Requires: bridge-utils
###################################

# Define Bridge Interface
br="br0"

# Define list of TAP interfaces to be bridged,
# for example tap="tap0 tap1 tap2".
tap="tap0"

eth_ip="192.168.8.4"
eth_netmask="255.255.255.0"
eth_broadcast="192.168.8.255"

for t in $tap; do
    openvpn --mktun --dev $t
done

brctl addbr $br

for t in $tap; do
```

```
    brctl addif $br $t
done

for t in $tap; do
    ifconfig $t 0.0.0.0 promisc up
done

ifconfig $br $eth_ip netmask $eth_netmask broadcast $eth_broadcast
```

Appendix B.

IDS Alerts generated for each test setup during the stability testing.

Test	Traditional Setup	i1	i2	i3	i4	i5	i6	
SQLi								
	2011042; Rev: 3	1	1	1	1	1	1	
	2010963: Rev: 4	1	1	1	1	1	1	
	2006609: Rev: 7	1	1	1	1	1	1	
	2006445: Rev: 10	1	1	1	1	1	1	
XSS	2009715; rev: 5	1	1	1	1	1	1	
RoR CVE-2013-0156 Scan								

	2016204: Rev: 3	1+	1+	1+	1+	1+	1+	
	2016175: Rev: 3	1+	1+	1+	1+	1+	1+	
	2016305: Rev: 6	1+	1+	1+	1+	1+	1+	
Remote/Local File Inclusion								
	2002997: Rev: 9	1+	1+	1+	1+	1+	1+	
	2013001: Rev: 4	1+	1+	1+	1+	1+	1+	
Test	**Local CC Setup**	**i1**	**i2**	**i3**	**i4**	**i5**	**i6**	
SQLi								
	2011042; Rev: 3	1	1	1	1	1	1	
	2010963: Rev: 4	1	1	1	1	1	1	
	2006609: Rev: 7	1	1	1	1	1	1	
	2006445: Rev: 10	1	1	1	1	1	1	
XSS	2009715; rev: 5	1	1	1	1	1	1	

RoR CVE-2013-0156 Scan								
	2016204: Rev: 3	1+	1+	1+	1+	1+	1+	
	2016175: Rev: 3	1+	1+	1+	1+	1+	1+	
	2016305: Rev: 6	1+	1+	1+	1+	1+	1+	
Remote/Local File Inclusion								
	2002997: Rev: 9	1	1	2	1	2	2	
	2013001: Rev: 4	1	0	1	1	1	1	
Test	**RS CC Setup**	**i1**	**i2**	**i3**	**i4**	**i5**	**i6**	
SQLi								
	2011042; Rev: 3	1	1	1	1	1	1	

	2010963: Rev: 4	1	1	1	1	1	1	
	2006609: Rev: 7	1	1	1	1	1	1	
	2006445: Rev: 10	1	1	1	1	1	1	
XSS	2009715; rev: 5	1	1	1	1	1	1	
RoR CVE-2013-0156 Scan								
	2016204: Rev: 3	1+	1+	1+	1+	1+	1+	
	2016175: Rev: 3	1+	1+	1+	1+	1+	1+	
	2016305: Rev: 6	1+	1+	1+	1+	1+	1+	
Remote/Local File Inclusion								
	2002997: Rev: 9	1+	1+	1+	1+	1+	1+	
	2013001: Rev: 4	1+	1+	1+	1+	1+	1+	