1-1-2012

Security Implications of the Cisco Nexus 1000V

Benjamin D. Peterson
bdpeters@purdue.edu

Graduate School ETD Form 9
(Revised 12/07)

PURDUE UNIVERSITY
GRADUATE SCHOOL
Thesis/Dissertation Acceptance

This is to certify that the thesis/dissertation prepared

By Benjamin D. Peterson

Entitled
Security Implications of the Nexus 1000V

For the degree of Master of Science

Is approved by the final examining committee:

Phillip T. Rawles
Chair

Michael J. Dyrenfurth

Raymond A. Hansen

To the best of my knowledge and as understood by the student in the *Research Integrity and Copyright Disclaimer (Graduate School Form 20)*, this thesis/dissertation adheres to the provisions of Purdue University's "Policy on Integrity in Research" and the use of copyrighted material.

Approved by Major Professor(s): Phillip T. Rawles

Approved by: Jeffrey L. Brewer 04/18/2012
Head of the Graduate Program Date

Graduate School Form 20
(Revised 9/10)

PURDUE UNIVERSITY
GRADUATE SCHOOL

Research Integrity and Copyright Disclaimer

Title of Thesis/Dissertation:

Security Implications of the Nexus 1000V

For the degree of Master of Science

I certify that in the preparation of this thesis, I have observed the provisions of *Purdue University Executive Memorandum No. C-22,* September 6, 1991, *Policy on Integrity in Research.**

Further, I certify that this work is free of plagiarism and all materials appearing in this thesis/dissertation have been properly quoted and attributed.

I certify that all copyrighted material incorporated into this thesis/dissertation is in compliance with the United States' copyright law and that I have received written permission from the copyright owners for my use of their work, which is beyond the scope of the law. I agree to indemnify and save harmless Purdue University from any and all claims that may be asserted or that may arise from any copyright violation.

Benjamin D. Peterson

Printed Name and Signature of Candidate

4/18/2012

Date (month/day/year)

*Located at http://www.purdue.edu/policies/pages/teach_res_outreach/c_22.html

SECURITY IMPLICATIONS OF THE CISCO NEXUS 1000V

A Thesis

Submitted to the Faculty

of

Purdue University

by

Benjamin D Peterson

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science

May 2012

Purdue University

West Lafayette, Indiana

ACKNOWLEDGMENTS

First, I would like to thank all of the faculty that I have worked with at Dublin Institute of Technology, Purdue University, and Universitat Politècnica de Catalunya. I am extremely grateful for all of the guidance and academic support that I have received throughout my time at each of the universities. In particular, I would like to thank Professor Nin and Rawles for their guidance during this research. I am equally grateful for the faculty that has made the Atlantis program possible. Having the opportunity to study in three different countries has undoubtedly been the experience of a lifetime. Last but certainly not least, I would like to thank my family and friends for providing me with the love and support throughout my studies.

TABLE OF CONTENTS

Page

Page

APPENDICES

LIST OF TABLES

LIST OF FIGURES

Figure Page

Figure Page

Figure Page

ABSTRACT

Peterson, Benjamin D. M.S., Purdue University, May 2012. Security Implications of the Cisco Nexus 1000V. Major Professor: Phillip Rawles.

Virtualization is a technology that has become increasingly popular with those wishing to reduce the energy consumption of their datacenters. This is especially true since virtualization technology allows multiple physical servers to be consolidated onto a single physical server in the form of virtual machines. Virtual networking devices have been created to allow these virtual machines to communicate amongst each other and with outside networks. Initially these virtual networking devices were crude; however, partnerships such as the one between Cisco and VMware have led to products such as the Nexus 1000V that have improved this network functionality. Despite the creation of the Nexus 1000V, the security implications of using the virtual switch have remained unclear. This research aimed to solve this. The outcomes of this research included tests of vulnerabilities previously or currently found on physical switches, an analysis of the communications used by the Nexus 1000V to support distributed switching, and an analysis of the effects of the switch existing as a virtual machine.

CHAPTER 1. INTRODUCTION

Virtualization technologies have been used for decades. In recent years it has been used to reduce the energy consumption of datacenters. This has been possible because the technology has been used to consolidate multiple servers onto fewer servers, with the consolidated servers being known as virtual machines. The resulting consolidation allows datacenters to reduce their number of physical machines therefore providing a significant reduction in the power necessary for running and cooling the servers.

In order to facilitate communication between the virtual machines, it has been necessary to provide the virtual devices with network functionality. Although initially primitive, this network functionality eventually evolved until the virtual switches mimicked the basic functionality of traditional network switches. Despite allowing the virtual machines to communicate with each other and outside of the virtual realm, these virtual switches did not provide the same functionality as physical switches. To close this gap, Cisco and VMware teamed up to create the Cisco Nexus 1000V. The touted benefits of the Nexus 1000V, ranging from improved networking functionality to more flexible management, were clear; however, the security implications of its use were not as clear.

The goal of this research was to determine the security ramifications of using the Nexus 1000V. It analyzed whether the vulnerabilities found in physical switches persisted into the virtual environment and determined whether it introduced new vulnerabilities.

An architecture utilizing the Nexus 1000V was created and its vulnerabilities were assessed. This architecture consisted of five servers running VMware's virtualization software ESXi. One of these servers was used to host VMware's virtualization management software, vCenter. The other four servers running ESXi had virtual machines installed on them as well the Nexus 1000V. All five of the servers were

connected to each other to allow communication amongst them. A sixth server was then used to run the traffic capturing software tcpdump.

1.1. Statement of the Problem

The Nexus 1000V has brought features previously only found in physical switches into the virtual realm. To accomplish this, some of the core functionality of physical switches was modified to better fit within the virtual environment. With these changes, there has been the potential for security issues of the virtual switches to be eliminated. Conversely, it has also been possible for new security issues unique to the Nexus 1000V to have been introduced. This research attempted to answer what the security implications of using a Nexus 1000V were.

1.2. Significance of the Problem

As companies have looked to increase the sustainability of their IT departments, many have turned to virtualization technology. Such virtualization technology has allowed for massive consolidation of servers, ultimately helping companies to reduce their hardware expenditures, energy usage, and maintenance. Traditionally, the technologies used to facilitate the virtualization of the servers have required the use of virtual switches that lack a majority of the functionality provided by traditional physical switches. However, Cisco's collaboration with VMware changed this. Together they created the Cisco Nexus 1000V switch. This virtual switch aimed to bring the functionality found in Cisco's physical switches to the virtual environments created by VMware's products.

Although the Nexus 1000V provides a multitude of features for its users, it was imperative that the security implications of its adoption were considered. As with all new networking equipment, it was vital that newly introduced security issues be determined. It is also important to determine which security vulnerabilities found in physical switches persist, and could now be found in the Nexus 1000V.

Failure to recognize and react to these security implications could not only put the integrity of the virtual networks at risk but it could also leave businesses facing significant financial losses. Intentionally triggered or not, an unrecognized security vulnerability in the new switch could leave the virtual networks crippled, rendering virtual servers unreachable. If a virtual switch was not properly protecting the data that it it handled and protected information reached an unintended party, businesses could face civil lawsuits and be at risk of regulatory incompliance. Without proper attention to these security aspects, there is a significant risk of a dramatic reduction or elimination of the return on investment for this technology.

1.3. Statement of the Purpose

The purpose of this study was to examine the functionality of the Cisco Nexus 1000V to determine the security implications of its use. This was to help determine whether Cisco's transition from physical switches to virtual switches had brought with it the inherent vulnerabilities of physical switches as well as whether or not it introduced new vulnerabilities unique to the virtual environment.

1.4. Definitions

This section defines the key terminology used throughout the research:

- Host: A server that runs virtualization software that facilitates the hosting of virtual machines on itself (VMware, 2009).
- Hypervisor: The software platform that runs on a host, that allows multiple operating systems to run at once (VMware, 2009).
- vCenter: A server platform made by VMware, that serves as a single point of management for the VMware virtual enviorment (VMware, 2009).
- Virtual Distributed Switch (vDS): A virtual switch that resides on multiple hosts and is configured from a single management point (VMware, 2009).
- Virtual Machine (VM): "A software computer that, like a physical computer, runs an operating system and applications" (VMware, 2009).

- Virtual Network Interface Card (vNIC): The virtual piece of hardware present on virtual machines, that allows the virtual machines to connect to vSwitches and pass network traffic (Cisco, 2009).
- VMware ESXi: Virtualization software made by VMware. VMware ESXi provides virtual machines with the environment and resources necessary to run (VMware, 2009).
- vSwitch: A software based switch that resides within the hypervisor and provides switching functionality within the virtual environment (Cisco, 2009).

1.5. Assumptions

The assumptions for this research were:

- The security of a single architecture with two distributed Cisco Nexus 1000V was representative of the security of all architectures involving distributed Nexus 1000Vs.
- An attacker could gain access to a virtual machine connected to a Nexus 1000V and would carry out attacks against it.
- The Nexus 1000V would have its default security features left it place.
- No security features existed on the virtual machines to prevent an attacker from manipulating the Nexus 1000V.
- If the Nexus 1000V were to use a form of Spanning Tree Protocol, it would be either the original Spanning Tree Protocol or Per VLAN Spanning Tree Protocol Plus.
- The features enabled by default on the Nexus 1000V represented the core functionality used.
- Datacenters would use 802.1Q VLANs to isolate the different groups of virtual machines.

1.6. Limitation

The limitations of this research were:

- The research was limited to the virtual environment and the physical switch to which the servers were connected.
- The research was focused on the security implications introduced by the use of the Nexus 1000V.

1.7. Delimitations

The delimitations for this project were:

- The research did not look into weaknesses in the encryption used to protect communications.
- The research did not focus on how an attacker might compromise a virtual machine from which it could launch attacks against the Nexus 1000V.

CHAPTER 2. LITERATURE REVIEW

Research covering the security of distributed virtual switches (dvSwitches), specifically the security of the Cisco Nexus 1000V, was rather limited at the time of this research; however, the topics that led to the creation of dvSwitches and the Cisco Nexus 1000V had been researched extensively. It was possible to group the precursor topics of the security of dvSwitches and the Cisco Nexus 1000V into four main categories:

- Switching
- Switch Vulnerabilities
- Virtualization
- Virtual Networking

In an attempt to discover thorough sources of information, several media were utilized. This media consisted of traditional books, product literature, and electronic databases. Topics extensively covered for many years were primarily researched using books. Information related to the Cisco Nexus 1000V and the specific virtual environments that supported it, was researched through the reading of product literature. Electronic databases were used to find information on topics that were still relatively current and had been covered in detail. The electronic databases used ranged from ones specific to computer related topics to others that were topic neutral. The predominantly used databases consisted of:

- Association of Computer Computing Machinery (ACM) Digital Library
- Compendex
- Institute of Electrical and Electronics Engineers (IEEE) Xplore
- Inspec
- Google Scholar

2.1. Switching

Switching has had an integral role in allowing electronic communication between two parties. In the first forms of electronic communications, a dedicated direct connection between the two communicating parties was necessary. With the increased usage of electronic communications, it was necessary to find a method that provided parties with a way to communicate with each other without requiring dedicated direct connections (Goldman & Rawles, 2004). The primary goal behind switching was to eliminate the need for dedicated direct connections by creating some form of path that allowed the electronic communication to reach its intended destination. Switching can be categorized into two different types, circuit switching and packet switching.

Circuit switching involves a dedicated path being temporarily created to facilitate the communications taking place. Prior to the electronic transmission beginning, the path is created. Soon after the communications have ceased, the created connection can be deconstructed. During the transmission, all of the circuit's resources are dedicated to the parties involved. The classic example for this type of switching is the public switched telephone network (PSTN).

Packet switching involves the use of many shared communication lines, over which the information is directed to its destination. These shared communication lines are referred to as networks. With this type of switching, the electronic communications are specially crafted and referred to as packets. As defined by Goldman and Rawles (2004), "Packets are specially structured groups of data that include control and address information in addition to the data itself" (p. 60). Devices that connect the networks to each other determine the path the communication travels. These devices use control and address information that is contained within the packet to determine the path it travels to reach its destination. To help in this decision, the devices take into consideration the current state of the network. Because of this, packets from the same source will not always take the same path to a destination. When communications are taking place on packet switched networks, the parties involved in communications have to share the resources of the network with all other parties using the network. The Internet is an example of a packet switched network.

One of the devices used to make decisions in packet switched networks is the appropriately named, switch. Switches use the information contained within packets to direct communications to their destinations. The information contained in these packets is determined by the protocols and frame standards used by the networks. For the sake of this paper, it is assumed that all network traffic uses Ethernet frames and the TCP/IP protocols. While other frame standards and protocols exist, Ethernet and TCP/IP are currently the most commonly used. The process switches use to make decisions is known as switching. There are four primary types of switching (Froom, Sivasubramanian, Frahim, & Houston, 2007). These types of switching are:

- Layer 2
- Layer 3
- Layer 4
- Layer 7

In layer 2 switching, switches analyze the destination MAC address that is contained within the packet's "header", a subset of the packet, to determine where to send it next (Froom, et al., 2007). This MAC address serves as a unique identifier for devices that are connected to the network. Network traffic contains a source MAC address for listing where traffic originated from, as well as a destination MAC address that denotes where the traffic is destined. Layer 2 switches makes their decisions by looking at the destination MAC address and comparing it with their content addressable memory (CAM) table. CAM tables contain information about the devices connected to the switch. This information is typically the MAC addresses associated with each port, along with the port's virtual local area network (VLAN) information (VLANs will be explained in the following paragraph). Switches populate the MAC address information by analyzing traffic's source MAC address. When a switch detects a new source MAC address, it will add it to the CAM table. These entries are not permanent and switches are typically configured to have the entries expire after five minutes of the switch not seeing the MAC address. If traffic is destined to a MAC address that is not in the CAM table, the switch will pass the traffic to all of the devices connected to it.

VLANs are logical networks that are used to segment networks. Network devices have the ability to send out a type of message known as a broadcast message (Farrow, 2003). On a switch without VLANs, if a device sends out a broadcast message, all of the devices attached to the switch receive the broadcast message. This can create a significant amount of network traffic when there are many devices connected to the switch. The purpose of VLANs is to breakup broadcast domains. Without VLANs, all devices on a switch would be in the same broadcast domain. There are two main protocols used for VLANs, Cisco Inter-Switch Link (ISL) and 802.1Q (Bastien, Nasseh, & Degu, 2006). These protocols require VLAN traffic to have tags that specify the VLAN to which it belongs. Switches are often configured to extend their VLANs to other switches; this is accomplished using trunk ports. Trunk ports allow tagged VLAN traffic to be sent between switches. To help facilitate the creation of trunk ports, Cisco developed a protocol known as Dynamic Trunk Protocol (DTP). Switch ports configured with this protocol detect whether connecting devices supports trunking. DTP can then be configured to dynamically configure the switch port as a trunk port, giving the connected device access to all of the VLANs.

IP addresses are used to make decisions in layer 3 switching. Just as layer 2 switches analyzed the MAC addresses of traffic to make their decisions, layer 3 switches analyze IP addresses to determine where to send packets (Froom, et al., 2007). Layer 3 switches ultimately make their decisions based off routing tables. In order to be aware of the IP addresses connected to other layer 3 devices, switches often employ routing protocols such as Routing Information Protocol (RIP), Enhanced Interior Gateway Routing Protocol (EIGRP), and Open Shortest Path First (OSPF).

Layer 4 and layer 7 switching are very similar to each other. Their decisions are determined by data in the packet that reflects what the packet is going to be used for instead of where it is destined (Froom, et al., 2007). The usage of this type of switching is typically application specific. For instance, it is possible to use layer 7 switching to determine whether a voice over internet protocol (VoIP) phone call is destined for a long distance phone number or a local number. With this information, the switch would be able to direct the traffic accordingly.

2.2. Switch Vulnerabilities

Throughout time, people have attempted to gain access to information and resources not intended for them. Networks have been no exception to this. Unfortunately, there have been vulnerabilities found in the functioning of switches that make them susceptible to this (Bastien, et al., 2006). These vulnerabilities have made it possible for attackers to compromise networks. It is possible to group the most widely exploited vulnerabilities into six categories. These categories are:

- CAM Overflow Attacks
- VLAN Hopping
- Spanning Tree Protocol Manipulation
- Address Resolution Protocol Poisoning
- Private VLAN Attack
- Dynamic Host Configuration Protocol Starvation

CAM overflows allow an attacker to take advantage of the CAM tables that were previously discussed. As was stated earlier, CAM tables record the MAC addresses associated with the switch's ports. It is important to note that these tables have a finite capacity (Bastien, et al., 2006). In this attack, an attacker floods the switch with traffic containing falsified MAC addresses, eventually filling the entire CAM table. Once the CAM table is full, the switch is unable to learn new MAC addresses. Because of this, the switches are forced to transmit traffic destined for unknown MAC addresses out all of its interfaces until there is free space in the CAM table. This in turn, presents the attacker with network traffic not intended for it.

VLANs are often implemented to provide isolation between different groups of devices (Farrow, 2003). Attackers are able to use a technique known as VLAN hopping to gain access to networks meant to be isolated from the network the attacker resides on (Bastien, et al., 2006). Attackers primarily use two techniques for carrying out this type of attack.

The first method involves an attacker taking advantage of a switch that has DTP configured on it (Bastien, et al., 2006). To achieve this, the attacker uses software that allows their computer to communicate as though it were a switch wishing to connect via

a trunk port. The attacker then has the software attempt to setup a trunk port with the switch to which it is connected. This tricks the victim's switch into creating a trunk port, giving the attacker full access to all of the VLANs on the victims switch.

The second method involves a technique known as "double tagging" (Bastien, et al., 2006). Double tagging is when an attacker creates a packet that contains two 802.1q tags, one with the VLAN that the attacker is supposed to be on and the other with the VLAN the attacker is wishing to reach. The packet is then sent to the switch it is attached to, the switch removes the first tag and then passes it on to the next switch. This second switch checks the packet, sees the second tag that was not removed, assumes that it is legitimate, and forwards the packet to the attacker's intended destination. This attack bypasses any restrictions that may have been in place between the two VLANs. It is important to note that this type of attack is unidirectional, as the contacted machine will continue to be affected by the restrictions that the attacker bypassed.

Attackers presently use the two mentioned attacks; however, another method existed during the early implementations of VLANs (Farrow, 2003). This attack took advantage of the fact that VLANs prevented other VLANs from being aware of the MAC addresses that resided within. At the time, VLANs offered no other form of isolation. In this method, an attacker that was aware of the MAC address of the machine it wished to communicate with, could still communicate with it by manually specifying the MAC address. As VLANs matured, this vulnerability was resolved.

Spanning Tree Protocol (STP) is a protocol used to prevent loops within switched environments (Bastien, et al., 2006). It is used when multiple switches are connected to each other. In brief, the protocol works by having the switches elect a "root" switch. This election is based off which switch has the lowest priority. This priority is can be manually configured. If two devices have the same priority value assigned to them, the devices MAC addresses are you to determine the root switch. Once a root switch has been determined, all of the other switches determine a path to it. This allows the switches to identify redundant connections that need to be shutdown to prevent loops. During the course of this process, the network is unusable. An attacker can manipulate STP by tricking the switches into believing the attacker is wishing to partake in the root switch

election process. When communicating with the switches, the attacker uses a priority that causes the switches to determine a new root switch. This then causes the network to be unusable as the switches re-determine redundant links. It is important to note that there are different versions of STP. In fact, Cisco has created their own version of it known as PVST+ (Vyncke & Paggen, 2008). Although it is a different version, it is still susceptible to such manipulations

Address Resolution Protocol (ARP) is a protocol used to relate IP addresses to MAC addresses (Bruschi, Ornaghi, & Rosti, 2003). When a host does not know the MAC address for an IP address, it broadcasts an ARP request. This request broadcasts to all hosts on the same IP subnet and requests the MAC address associated with the IP address. The other hosts on the subnet that receive it will check their own IP address to see if it is the one being requested. If the IP address is not associated with the host, the host will ignore the request. However, if the host is associated with the requested IP address, it will respond to the requester with an ARP reply. This reply will inform the requester of its MAC address. On most operating systems, the received address is cached on the requesting host to eliminate the need for repeating this process later. Another type of ARP message is a gratuitous ARP (Bastien, et al., 2006, 292). This type of message is sent out by a host to announce their IP address to the other devices on the network. In doing so, machines accepting gratuitous ARP messages will record the announced address information. Attackers are able to manipulate machines on the network by sending out spoofed ARP messages in a method known as "ARP poisoning" (Bruschi, et al., 2003). ARP poisoning consists of the attacker sending out spoofed ARP replies to hosts that they wish to manipulate. This causes the compromised machines to send traffic meant for another host to a host of the attacker's choosing.

Private VLAN attacks are similar to VLAN hopping, except they do not involve taking advantage of vulnerabilities in the VLAN protocols. Like the double tagging method of VLAN hopping, it facilitates a one-way communication with a machine that is on a protected VLAN (Bastien, et al., 2006). To carry out this type of attack, the attacker spoofs a packet so that it contains the destination IP address of the machine they are wishing to communicate with and a destination MAC address of a router connected to the

switch to which the attacker is connected. When the switch passes the packet to the router, the router then updates the destination MAC address to reflect the victim machine's MAC address and then forwards it onto the victim machine. The packet is able to bypass restrictions imposed by the router since the router is tricked into believing it sent the packet.

Dynamic Host Configuration Protocol (DHCP) is used to automatically send network configuration information, such as IP addresses allocations, to network devices connecting to a network (Bastien, et al., 2006). DHCP servers have a finite number of addresses that can be allocated to hosts. An attacker can take advantage of DHCP servers by flooding the server with spoofed DHCP requests, eventually causing the DHCP server to run out of addresses. Once the DHCP server has run out of addresses, devices that do not have addresses manually assigned to them will not be able to access the network. In addition, an attacker could run a rogue DHCP server and send network configurations that cause the victims to send their network traffic to the attacker instead of their intended destinations.

2.3. Virtualization

Virtualization is a technology that facilitates the abstraction of a computer's hardware allowing them to run software they were not designed for and to host other independent computers, known as virtual machines (VM). The use of virtualization technology dates back to the 1960s (Nanda & Chieueh, 2005; Smith & Nair, 2005). When virtualization technologies were first created, they were meant to provide time sharing and resource sharing, as well as allow multiple operating systems to be installed. During the 1970s and 1980s, computer hardware became cheaper, causing less of a demand for virtualization technologies (Nanda, 2005). In the 1990s they became popular once again but this time they were used for a variety of new purposes such as power saving, server consolidation, application consolidation, and debugging (Nanda, 2005; VMware, 2009). With the wide variety of purposes, many different types of virtualization technologies emerged. For the sake of maintaining a manageable paper

length, only virtualization at the Hardware Abstraction Layer (HAL) will be covered, as it is most applicable to the research that was carried out.

Virtualization taking place at the HAL allows for software known as a Virtual Machine Monitor (VMM) to run on a host's base hardware, much like an operating system would (Nanda, 2005). VMMs are often referred to as hypervisors (VMware, 2009). The VMM is used to map the host's physical resources to the virtual environment, allowing them to be allocated to and used by the VMs (Nanda, 2005). The ultimate goal of the VMM is to create "… a complete, persistent system environment that supports an operating system along with its many users processes" (Smith, 2005, p. 34). To aid in this, resources can be allocated using two different methods. The first method that can be used is known as physical partitioning. For this method, the host's physical resources are assigned and dedicated to a specific VM. The second method that can be used is known as logical partitioning. This method allows for the sharing of host's resources amongst the VMs. VMware's ESX and ESXi are both examples of virtualization technologies that run at the HAL (Nanda, 2005).

As companies have turned to virtualization, many have adopted VMware's ESX and ESXi products. Both of these solutions provide a virtualization layer that runs on the hosts and provides abstraction of resources into the virtual environment (VMware, 2009). The main difference between ESX and ESXi is that ESX has a built in service console. While ESXi lacks a service console, it can be embedded into a server's firmware. In order to provide more features and management capabilities to groups with datacenters running ESX, ESXi, or both, VMware created vSphere.

VMware states "vSphere virtualizes and aggregates the underlying physical hardware resources across systems and provides pools of virtual resources to the datacenter" (VMware, 2009, p. 7). It is important to note that ESX and ESXi still play a fundamental role in this solution. Both provide virtual environments to which vSphere interfaces. vSphere consists of four component layers. The first component layer is infrastructure services. This layer is responsible for providing the ability to share resources, storage and network capabilities amongst the hosts. The second component layer is the application services layer, which is responsible for ensuring high availability,

security, and scalability. The VMware vCenter Server component layer is responsible for providing management functionality for the virtual environment. The final component layer is the clients layer that is made up of the clients that interact with the virtual environment.

vSphere offers its users a multitude of features for improving their datacenters. vMotion allows VMs to be moved from one host to another without affecting the availability of the VM (VMware, 2009). The Distributed Resource Scheduler (DRS) and Distributed Power Manager (DPM), allows datacenters to reduce their power consumption (VMware, 2009). To accomplish this, the DRS dynamically allocates resources to the VMs, as they are needed. The DPM analyzes the hosts that are in use and determines whether VMs could be consolidated to a fewer number of hosts. If it is determined, that host consolidation is possible, the VMs are moved, and the unnecessary hosts are powered down.

2.4. Virtual Networking

With multiple VMs on a single host, it was necessary to establish a way for the VMs to communicate effectively with each other as well as outside of the virtual environment. To facilitate network communications, VMs were provided with a virtual network interface card (vNIC) that allowed them to transmit network communications (Zhou, 2010). Initially VM's vNICs were connected via virtual hubs. The use of virtual hubs caused the VMs to see all network traffic going through the hub regardless of the traffic's destination. As the number of VMs on a host increased, the virtual hubs became a performance bottleneck. Software based network bridges was introduced in the late 1990s (Pettit, Gross, Pfaff, Casado, & Crosby, 2010). Virtual switches that provided layer 2 switching followed soon after. The virtual switches integrated into the VMMs and used the host's resources, just as the VMs did (Luo, Murray, & Ficarra, 2010). Originally, these virtual switches lacked management features (Zhou, 2010). As they matured, they included a subset of features commonly found in physical switches. These included basic security features, VLANs, and portgroups. Portgroups provided a way in which administrators could create a standardized network configuration for their virtual

machines. This was accomplished by assigning network configurations to portgroups. Instead of administrators having to manually configure the network settings for each VM, a portgroup could be selected to apply the predetermined configuration associated with the portgroup.

Despite the improvements made to the network functionality in the virtual environment, there were still some unmet needs. It was still necessary for the repeated configuration of the virtual switches and portgroups on each host (Zhou, 2010). This proved to be especially daunting for ensuring consistent configurations across large datacenters. To overcome this issue, distributed virtual switches (dvSwitches) were developed.

The concept behind dvSwitches was to allow virtual switches to communicate with each other so that they could share state and configuration information with each other. One of the earliest examples, presented at a conference, allowed virtual switches to be aware of the MAC addresses associated with the virtual switches it was connected to (Davoli, 2005). This conceptual example consisted of two components, vde_switches, and vde_cables. The vde_switches were responsible for handling the basic switching functionality and the vde_cables were used to connect two vde_switches together. The vde_cables were made up of two components, vde_plugs, and an interconnection mechanism. Vde_plugs were added to both of the vde_switches that were to be connected. The purpose of the vde_plugs was to create a stream of information to be sent to the other switch. Very little restrictions were placed on what could serve as the interconnection mechanism. The only requirement was that the interconnection mechanism must allow transmission from one end to another. Examples of possible interconnection mechanisms included Secure Shell (SSH), Remote Shell (RSH) and SLIRP. It is important to note that this implementation was meant as a proof of concept and had limited practical applications.

As dvSwitches matured, they became more common in virtualization solutions. One of the main advantages to their use was that they provided a single point of management for multiple virtual switches that resided on multiple hosts (Zhou, 2010). Portgroups that were configured on dvSwitches were known as dvPortgroups. Another

important feature provided by dvSwitches was the concept of dvPorts. dvPorts allowed for a VM's network configuration to be moved with the VM. This helped facilitate the transition of VMs from one host to another without minimal service interruptions.

Despite the features provided by dvSwitches, many of features required by network security devices, such as intrusion detection systems, were still absent. In response to this and other issues, application programmable interfaces (APIs) for the virtualization technologies were created (Zhou, 2010). Such APIs allowed third parties to develop their own switching solutions that integrated into the virtual environment. This gave third parties the freedom to add their own features. Two current examples of technologies that utilize these APIs are the Open vSwitch (Pettit, et al., 2010) and Cisco Nexus 1000V (Zhou, 2010).

The Cisco Nexus 1000V was Cisco's first product to take advantage of the Cisco VN-Link technology (Cisco, 2010). The goal of VN-Link technology was to provide Cisco's switching features and configuration options to the virtual environments (Cisco, 2009). To help it integrate with the virtual environments, it aimed to "provide policy-based virtual-machine networking" (Cisco, 2010, p. 6). VN-Link technology was able to achieve this by utilizing the APIs in VMware's vSphere to integrate into the virtual environment's VMM.

Using Cisco's VN-Link technology, the Cisco Nexus 1000V is able to provide network functionality in the virtual environment. The virtual switch runs the Cisco's NX-OS operating system to provide an interface and configuration similar to Cisco's physical switches (Cisco, 2011b). Two key components, the virtual Ethernet modules (VEM) and the virtual supervisor modules (VSM) provide its functionality (Cisco, 2009, p. 6; Cisco, 2010, p. 7). The VEM is used for providing switching functionality, as well as other network functionality such as security features. The VSM allows for the management of the switch. It provides an interface that allows for the "… configuration, management, monitoring, and diagnostics of the overall system (that is, the combination of the VSM itself and all the VEMS that it controls) as well as integration with VMware vCenter" (Cisco, 2009; Cisco, 2010). The Cisco Nexus 1000V supports up to 64 VEMs and two VSMs spanned across 64 hosts (Cisco, 2011b). Each VEM allows for 512

VLANs, with each VLAN being able to support 1024 MAC addresses. Another important characteristic of the Cisco Nexus 1000V is that it does not use STP within the virtual environment; instead, it relies on specially designed forwarding logic to prevent loops in the network.

2.5. Summary

While little research existed on the security of dVSs and the Cisco Nexus 1000V, it was possible to find valuable insight into the precursors of these technologies. Research allowed for a historical and operational understanding of switches. Information on the vulnerabilities in switches served to highlight possible issues that may have been still present in the Cisco Nexus 1000V. Since the Cisco Nexus 1000V resides within virtual environment it was imperative to understand the technology used to facilitate the virtual environments. Finally, it was necessary to review as much literature as possible about the Cisco Nexus 1000V to gain an understanding of how it functions. In summary, while this research was not specific to the security implications of the use of the Cisco Nexus 1000V, this information provided a strong survey of the precursor topics that will serve as a solid foundation for this experiment.

CHAPTER 3. METHODOLOGY

This research was focused on determining the security implications of using the Nexus 1000V with respect to a traditional physical switch. In order to carry out this research it was necessary to break the research in to four distinct steps. The first step was to create an architecture in which the Nexus 1000V would reside. It was imperative that this architecture demonstrated the functionality used in real-world situations. The next step was to analyze the security of the virtual switch against vulnerabilities found or previously found in physical switches. To accomplish this, attack methods previously and currently used against switches were carried out against the Nexus 1000V to determine whether it was vulnerable to attack. After establishing which switching vulnerabilities persisted from the physical switching environment into the virtual switching environment, an analysis of the communications used by the Nexus 1000V to facilitate distributed switching were conducted. This analysis consisted of determining how the switches were able to communicate, an analysis of the communications, and an attempt at manipulating these communications. The final step in this research was to look at the effects of duplicating the Nexus 1000V virtual machines.

3.1. Preface

Prior to beginning the experiment, it was necessary to clarify the definition of a security implication. It was decided that a security implication would be any type of activity that caused the Nexus 1000V to act in way it was not intended. While any diversion from normal operation may seem drastic to deem a security implication, it is important to consider that vulnerabilities are often used as “stepping stones” for accomplishing objectives that are more nefarious.

It was determined that there were a variety of methods in which an attacker could have potentially affected the Nexus 1000V's standard operation. These were primarily methods that had been used against physical switches. Such vulnerabilities included:

- CAM overflows
- VLAN Hopping
- STP Manipulation
- ARP Poisoning
- Private VLAN Vulnerabilities

It was also possible that the Nexus 1000V introduced new vulnerabilities. In particular, it was possible that the means of communication used to facilitate distributed switching had vulnerabilities in it. There was also potential that the mere fact the Nexus VSM resided as a virtual machine could also pose as a security implication. Regardless of whether the potential vulnerabilities had previously existed in physical switches or were introduced with the transition to the virtual realm, each posed a potential security implication. A graphical depiction of the stated vulnerabilities is depicted in Figure 3.1.

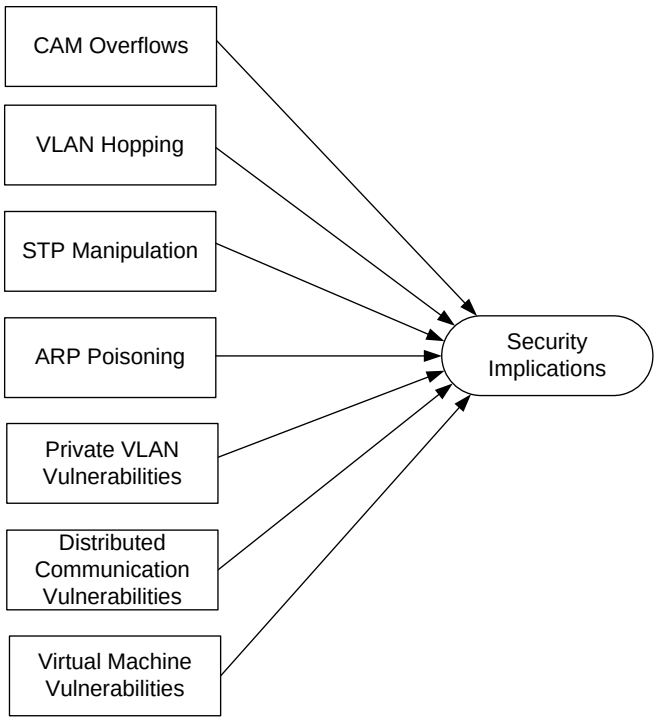

Figure 3.1. A visual depiction of the security implications.

3.2. Test Architecture

Before the analysis of the security implications of the Cisco Nexus 1000V could begin, it was imperative to establish a test architecture that accurately represented other implementations of the Nexus 1000V. Since any security implication affecting the core functionality of the Nexus 1000V would affect other implementations of the Nexus 1000V, only one sample was necessary. This sample however needed to provide the core functionality found in implementations of the Nexus 1000V. To accomplish this, it was first necessary to establish what functionality was utilized. While the Nexus 1000V is commonly implemented within datacenters, it would have been impossible to simulate a datacenter with the available resources. Therefore, it was necessary to determine a scaled down architecture that accurately represented the core functionality of the Nexus 1000V. The final step was to create a working implementation of the determined architecture.

The first step was to determine what functionality was commonly used in implementations of the Nexus 1000V. To make this determination it was necessary to first determine what virtualization software would be used. The Nexus 1000V is only supported by architectures running VMware's ESX or ESXi and vCenter (Cisco, 2011b). This made it necessary to have a virtual environment with ESX or ESXi servers being managed by vCenter. It was also realized that the research would be most relevant if the most current versions of the virtualization software was used.

The second step was to determine the common deployments of the Nexus 1000V. Since the Nexus 1000V was designed to be implemented in environments running VMware's vCenter, which was developed for use in datacenters (VMware, 2009), it was safe to assume that it would commonly be implemented in datacenters. With it being used in a datacenter environment, it was assumed that at least five servers hosting the virtualization software would be in use. Companies with datacenters often use some form of redundancy in an attempt to minimize downtime. In an attempt to provide this, it was determined that it would be likely that two servers would have VSMs on them and all of the other servers would have VEMs running on them. It was also assumed that implementations using the Nexus 1000V would have multiple VLANs to isolate the virtual machines.

With the typical implementation determined, it was possible to establish an architecture that represented the architectures commonly used in datacenters. The goal was to find the least number of servers necessary to provide the functionality commonly used with implementations involving the Nexus 1000V. It was determined that it would be necessary to have one server running vCenter to provide the management functionality for the virtual environment. The most current version of vCenter at the beginning of this research was vCenter Server 5 and it was therefore used. vCenter's standard license supported the necessary functionality and was used as well. The decision as to whether ESX or ESXi should be used was found to be indiscriminate, as both provided the same functionality with the only difference being how they can be managed (VMware, 2009). It was also decided that ESXi 5.0 would be used, as it was the most current version available at the start of the experiment. To allow the Nexus 1000V to be integrated into the virtual environment, it was necessary to utilize Enterprise Plus licenses. The number of necessary virtualization hosts was related to the number of VSMs and VEMs required. Although, it was possible to have one VSM, having only one would mean that the redundant supervisor functionality and VSM to VSM communications would be non-existent. With this, it was decided that it would be necessary to have two VSMs instead of one, as having only one would limit the areas this research explored. To simulate redundant VSMs, it was concluded that at least two of the virtualization servers needed to have virtual machines running the Nexus 1000V configured as a VSM (Cisco, 2011a). It was determined that it would be necessary to have at least two hosts, each with a VEM to provide VEM to VEM communications. Finally, in order to be able to monitor the communication between the ESXi hosts, a separate server with network monitoring software was deemed necessary.

Once the requirements were determined, it was possible to create the necessary virtual environment. To accomplish this, six servers and one physical switch were necessary. Due to the resources available, there was a heterogeneous variety of servers used. They were composed of a mixture of two Dell Optiplex 745s, two Dell Optiplex 755s, one Dell Optiplex 990, and a Dell Optiplex GX620. These servers were then connected to one another using a Cisco 3750-24TS-S running Cisco IOS 12.2. Each of

the servers had physical network connections to the network, one to be used for management of the server and the other for providing the server's intended functionality. Further detail on the specific hardware is available in Table 3.1 and the physical network is detailed in the diagram depicted in Figure 3.1.

Table 3.1.

Summary of Hardware Used in the Test Architecture

Machine	Model	Processor	Memory
ESXi10	Dell Optiplex 755	Intel Core 2 Duo E6750	4 GB
ESXi11	Dell Optiplex 745	Intel Core 2 Duo 6400	4 GB
ESXi12	Dell Optiplex 745	Intel Core 2 Duo 6400	4 GB
ESXi13	Dell Optiplex 755	Intel Core 2 Duo E6750	4 GB
ESXi14	Dell Optiplex 990	Intel Core i7-2600	8 GB
Sniffer	Dell Optiplex GX620	Intel Pentium D	2 GB

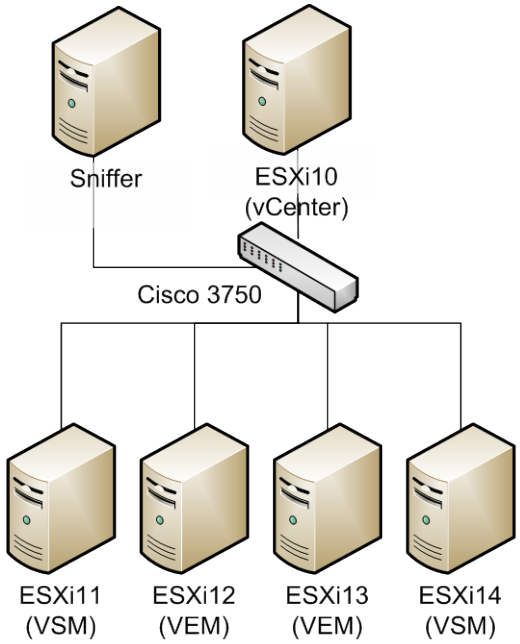

Figure 3.2. The physical server architecture used to represent typical implementations of the Nexus 1000V.

With the architecture's physical network connections in place, it was then necessary to configure the physical switch. The switch was configured so that each of the server's management interfaces was on VLAN 1935. Four additional VLANs were created to facilitate the Nexus 1000V's functionality for carrying out this security assessment. The created VLANs and their purpose are detailed in Table 3.2. Once the VLANs had been configured, a trunk port to ESXi10, 11, 12, 13, and 14 was created that carried each of the created VLANs. For further detail on the physical switch, refer to its configuration file in Appendix A.

Table 3.2.

VLANs Used in the Test Architecture

VLAN	Name	Purpose
971	Control	Internal Nexus 1000V Communication
972	Packet	Virtual machine network traffic
973	SecureVLAN	Virtual machine network traffic
974	PrivateVLAN	Isolated virtual machine network traffic
1935	Management	Management of the servers and virtual machines

After the physical switch had been configured, VMware's ESXi 5.0 was installed and configured on ESXi10, 11, 12, 13, and 14. It should be noted that in order to install ESXi on ESXi14, it was necessary to use a custom installation that included drivers for the Intel 82579LM network card. The ESXi10 host hosted a virtual machine running Windows Server 2008 R2 standard. On this virtual machine, vCenter 5.0 was installed and configured to manage the other ESXi hosts. Once vCenter had been configured to manage the ESXi hosts, each of the managed ESXi host's non-management network interfaces was configured with a vSphere Standard Switch that provided connectivity to the Control, Packet and Management VLANs.

On ESXi10 and ESXi14, a virtual machine running the Nexus 1000V configured as a VSM installed on it. The VSM was installed according to the steps detailed in the *Cisco Nexus 1000V Installation and Configuration Guide* (Cisco, 2012). Following this

guide, the VSM was installed using the Nexus 1000V Installation Management Center, which had been downloaded onto the virtual machine running Windows Server 2008. For this installation, ESXi14 was selected for the installation host. During the installation, "HA" (high availability) was selected for the redundancy mode. When configuring the port-groups, "L2" (layer two) was selected for the port-group type and the Control, Packet and Management port-groups were associated with their respective VLANs. After the VSM virtual machines had been created, the Nexus 1000V Installation Management Center was used to migrate the port groups and network adapters from the standard switch to the VSM. The final step of this process was to migrate the secondary VSM from the ESXi14 host to the ESXi10 host.

With the creation of the VSMs on the two ESXi hosts, the VEMs were then installed on ESXi11 and ESXi12. This was accomplished through using Nexus 1000V Installation Management Center. As with the installation of the VSM, the guidelines laid out in the *Cisco Nexus 1000V Installation and Configuration Guide* were followed (Cisco, 2012). Once VEM installation had completed, the installation was verified by ensuring that the port-groups associated with the port-profiles on the VSM were now available on the ESXi11 and ESXi12 hosts. After the Nexus 1000V's VSMs and VEMs had been successfully been deployed, the switch was configured so that the VLANs previously created on the physical switch were accessible through the Nexus 1000V. Next, port-profiles were created that allowed these VLANs to be assigned via the port-groups in vCenter. For further detail on the VSM's configuration, refer to the VSM configuration file in Appendix B.

With the Nexus 1000V deployed and configured, virtual machines were added to each of the hosts. Although, most datacenters would consist of a heterogeneous mixture of operating systems, it was decided that this would not be necessary. This determination was due to the research being focused on the functionality of the switch rather than underlying operating systems. Since the majority of network traffic adheres to standards regardless of the operating system, the operating system that was most appropriate for tests was used. In this case, virtual machines running CentOS 5.4 and Backtrack 5 were used. CentOS virtual machines were used to create legitimate network traffic and a

combination of CentOS and Backtrack virtual machines were used to carry out the security tests. A CentOS machine virtual machine for creating legitimate traffic was then placed on each of the ESXi hosts. The CentOS and Backtrack virtual machines were then added to the hosts as well. All of the virtual machines that were created were created with the "flexible" type of network interface. All of these machines contained only one network interface, except for the CentOS attack machines. These machines were configured with two network interfaces, with one being used for administering the virtual machine and the other being used for carrying out tests. As these virtual machines were created, they were placed into the Packet port-group that had been created during the installation of the Nexus 1000V. In addition to these virtual machines, an additional virtual machine running CentOS was added to each of the ESXi hosts. The purpose of these virtual machines was to capture network traffic between the virtual machines. To facilitate this, they were created with two network interfaces. One interface was placed into the Management port-group to allow for the administration of the virtual machine. The second interface was then added to the Packet port-group. It is important to note that throughout this research the port-groups and port-group configurations changed as noted in the experiments. It was also necessary to move the virtual machines amongst the hosts to facilitate some of the tests. This is also noted in the experiments. For further detail on the created virtual machines refer to Appendix C, which contains a VMX file for each type of virtual machine that was used and Appendix D, which contains the MAC addresses used by the virtual machines.

In order to allow for the analysis of network traffic between the virtualization servers, a server running CentOS 5.4 was used. This server came with the network monitoring software, tcpdump, installed on it, and was added to the same physical switch to which the others had been added. Like the virtual monitoring servers, this server also had two network interfaces, with one being used for management and the other for monitoring network traffic. For further detail on this and the previously mentioned machines refer to Figure 3.3, which depicts the logical network architecture that was used for this research.

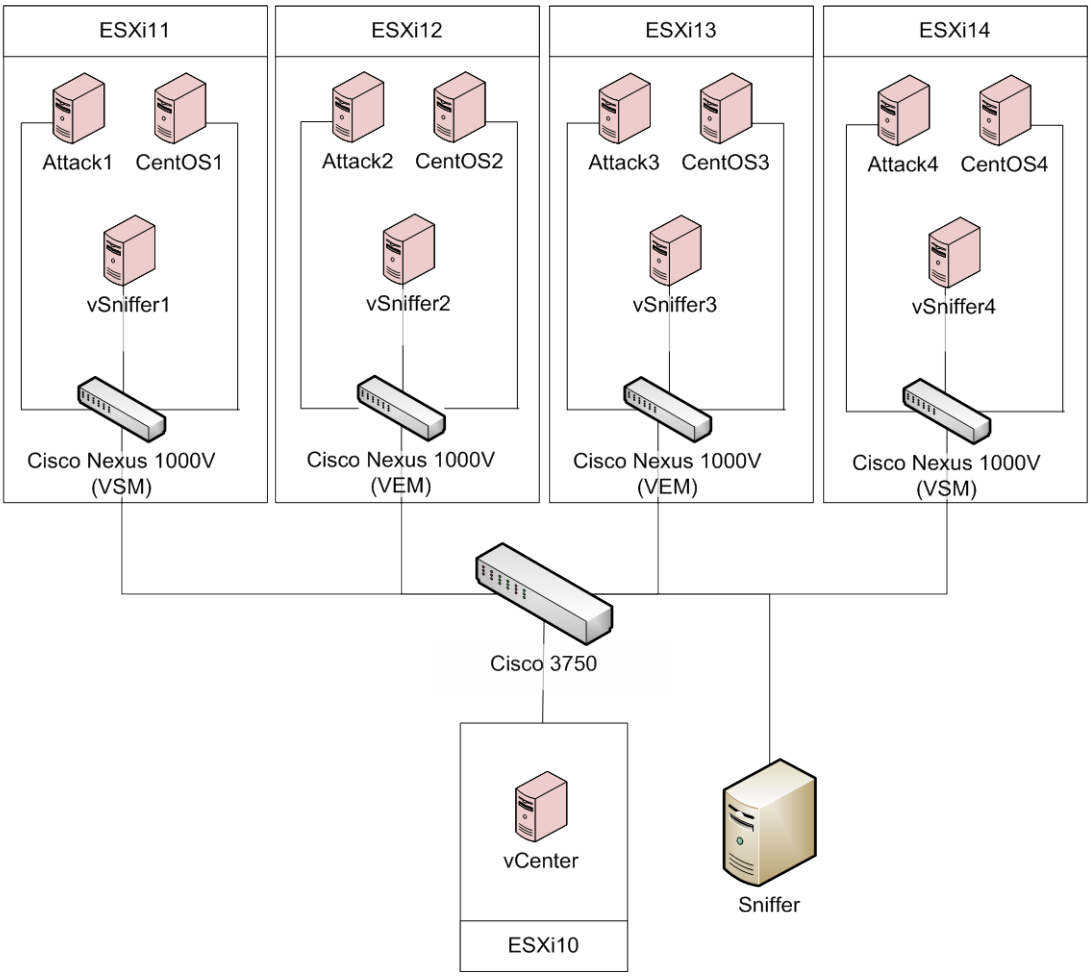

Figure 3.3. The logical architecture used for this research.

3.3. Physical Switch Vulnerabilities

To gain a perspective as to how the Cisco Nexus 1000V compares to physical switches, the previously stated vulnerabilities commonly found or previously found on physical switches were sought out on the virtual switch. To assess whether these vulnerabilities existed on the Nexus 1000V, it was necessary to use the same tools that were used to carry out the attacks on physical switches. Each of these vulnerabilities was tested from either a CentOS or Backtrack attack virtual machine that resided on the virtual networks being serviced by the Nexus 1000Vs.

CAM overflows were tested against the Nexus 1000V using the macof tool (Song, n.d.). This tool was designed specifically for testing switches' vulnerability to CAM overflows. To achieve this, the tool rapidly generated network traffic that contained forged MAC addresses. The switch to which the virtual machine running macof was connected was then forced to handle and interpret this crafted traffic. To test for this in this specific environment the command "macof -i eth1" was used on Attack2, the virtual machine running Backtrack on ESXi host ESXi12. This virtual machine was connected to the Nexus 1000V and placed on VLAN 972. In order to verify that legitimate traffic was impacted by this attack, ICMP traffic was generated between three of the CentOS hosts.

To test for VLAN hopping vulnerabilities, it was necessary to use two tools. Nemesis, a packet-crafting tool was used to create packets destined for MAC addresses on other VLANs (Nathan, n.d.). Yersinia was used to create packets with two 802.1Q headers in an attempt to carry out VLAN hopping (Omella & Berrueta, n.d.).

Nemesis was used on the Attack1 a virtual machine running CentOS that was placed on ESXi12. For this test, Attack1 was placed onto VLAN 974, which had had an access list preventing its devices from communicating with devices on VLAN 972. On the attack machine, Nemesis was used to create an ICMP request with the MAC address of the CentOS2 virtual machine that also resided on ESXi12 but was on VLAN 972. The command used to create the ICMP request was "nemesis icmp -D 10.97.2.102 -M 00:50:56:AB:56:3C -B 10.97.4.102".

Yersinia was used on the same CentOS attack virtual machine as the used in the first VLAN hopping test; however, the port-profile for this virtual machine was updated so that the switch port it was connected to was on VLAN 1, the native VLAN. It is important to note that the switch port was still configured in access mode. For this test Yersinia was run in interactive mode by entering the command "yersinia –I". Once in interactive mode, the protocol mode was set to "DTP" and the "Trunking" attack was executed. With the "Trunking" attack continuing to execute, the protocol mode was switched to "802.1Q". In the 802.1Q mode, the packet configuration was configured so that its initial 802.1Q header tagged it as VLAN 1 and the second header had the packet tagged as VLAN 972. In doing so, the header containing VLAN 1 would be removed but

the VLAN 972 tag would persist. Figure 3.4 depicts the updated configuration dialog that was used. After the configuration was set, the “802.1Q Double Encoded Packet” attack was executed.

```
Source MAC 0E:5C:49:19:32:BF Destination MAC 00:13:C3:9D:0B:C4
VLAN 0001 Priority 07 CFI 00 L2Proto1 0800 VLAN2 0972 Priority 07 CFI 00
L2Proto2 0800 Src IP 010.097.002.166 Dst IP 010.097.002.002 IP Prot 01
Payload YERSINIA
```

Figure 3.4. The Yersinia configuration used to test for VLAN hopping.

Although the switch utilized an algorithm that was meant to eliminate the need for STP, there was potential that it would still process the STP communication in some manner. To check its vulnerability, falsified STP communications were sent from the virtual machines to the Nexus 1000V. Yersinia was once again used, as it allowed for the creation of falsified STP packets (Omella, n.d.). In particular, it was used to test whether the switch was vulnerable to manipulation from STP packets matching the original STP standard. Because Yersinia did not support the sending of PVST+ STP packets, it was necessary to create these packets using Nemesis.

For the first test, Yersinia was used to create falsified STP packets meeting the original STP standard. For this test, the CentOS attack machine previously used was placed on VLAN 972 and two STP manipulations were attempted. The first manipulation was a Configuration BPDU denial-of-service and the second was a Topology Change Notification BPDU denial-of-service. Both of these attempts were accomplished by again using Yersinia’s interactive mode. This time however, the “STP” attack group was selected. The configuration for these packets was based on the information detailed by Vyncke and Paggen (Vyncke, 2008). Figure 3.5 reflects the updated configuration dialog that was used. After the configuration had been updated, the two tests were carried out.

```
Source MAC 04:08:20:12:A9:75 Destination MAC 01:80:C2:00:00:00
Id 0000 Ver 02 Type 00 Flags 40 RootId 0058.E7CD90117CAA Pathcost 00000000
BridgeId 0058.E7CD90117CAA Port F381 Age 0000 Max 0014 Hello 0002 Fwd 000F
```

Figure 3.5. The Yersinia configuration used to test for STP vulnerabilities.

For the second attempt at STP manipulation, it was necessary to use Nemesis to create PVST+ packets. This was accomplished by capturing a PVST+ packet that had been sent by the physical switch and modifying it. Once the PVST+ packet had been captured, it was analyzed in hexadecimal form. First, its Ethernet header was removed and then the BPDU type was updated to "80", so that it would be classified as a topology change notification. The Topology Change Notification bit was also incremented from "0" to "1", changing the BPDU flag from "3c" to "3d". Once the payload had been updated, it was necessary to convert the payload from hexadecimal to binary. This was accomplished by using a perl script named hex2bin (B4rtm4n, 2005). With the payload converted to binary, it was possible to send the payload using Nemesis; however, since Nemesis did not offer a feature that assigned a random source MAC address, it was necessary to create a script that did so. This was accomplished by creating a BASH shell script that generated a random MAC address and used it as the source parameter while executing Nemesis. In order to generate the random MAC address, code by the internet user Vaporub was used (Vaporub, 2009). With the source address being randomly generated, "01:00:0c:cc:cc:cd" was used for the destination MAC address, as this the destination address used by PVST+ (Vyncke, 2008). The final step in the creation of the script was to have it continuously loop, causing the switch to be flooded with PVST+ BPDUs from random MAC addresses. Figure 3.6 depicts the script that was used for this test. After the Topology Notification Change test had completed, the payload was updated so that the packet type was "00" and would be interpreted as a Configuration BPDU. Following the updates, the Configuration BPDU test was carried out.

```
#!/bin/bash
COUNTER=0
while [ $COUNTER -lt 1 ]; do
MAC=$(echo `(date; cat /proc/interrupts) | md5sum | sed -r 's/^(.{10}).*$/\1/; s/([0-9a-f]{2})/\1:/g; s/:$//;'`)
echo $MAC
nemesis ethernet -v -d eth0 -H $MAC -M 01:00:0c:cc:cc:cd -T 33024 -P /wrk/payload_tcn.bin
```

Figure 3.6. The script used for testing for PVST+ vulnerabilities.

Testing the Nexus 1000V's vulnerability to ARP poisoning was possible through the use of a tool known as ettercap. It was found that it was possible to use this tool to create falsified ARP responses that were sent to the other virtual machines (Ornaghi &

Valleri, 2005). Ettercap was used from the same Backtrack attack virtual machine that had been previously used for the CAM overflows. It was also placed on the 972 VLAN, the same VLAN to which the other CentOS hosts were connected. To test the vulnerability to ARP poisoning, ettercap was used in an attempt to cause the other virtual machines on the 972 VLAN to direct network traffic to the attack virtual machine. The specific command used to carry out this test was "ettercap -i eth1 -T -q -M arp:remote /10.97.2.2-110/ -P autoadd". To generate traffic that was to be intercepted, ICMP communications were sent amongst the CentOS hosts that resided on VLAN 972.

Nemesis was used to find whether the Nexus 1000V was susceptible to private VLAN attacks (Nathan, n.d.). Using Nemesis, packets were created with the destination MAC address of the gateway that the virtual machine was using (Nathan, n.d.). For this experiment, the CentOS hosts were moved to VLAN 974, which had been configured as a private VLAN for this experiment. In particular, this VLAN had been configured in isolation mode so that none of the hosts would be able to communicate with each other. To enable private VLANs, it was necessary to issue several commands. It was ultimately necessary to reload the switch with the updated configuration. For further detail on the configuration used, refer to the configuration file in Appendix E. Once on the private VLAN, it was verified that the hosts could not communicate with each other. After this had been verified, Attack1, the attack virtual machine running CentOS, was also added to the private VLAN. This virtual machine was then used in an attempt to send packets to the legitimate CentOS hosts. To do this it was necessary to use Nemesis to create a custom ICMP packet that had the attacking virtual machines source IP address and the intended destinations IP address. The critical part of this packet was to have the destination MAC address be that of the default gateway. It was hoped that doing so would push the packet to the default gateway, which would then forward the packet to the supposedly protected CentOS host. To accomplish this with Nemesis, the following was used "nemesis icmp -d eth0 -S 10.97.4.66 -D 10.97.4.102 -M 00:1c:0f:5c:00:40".

To verify the impacts of these tests a variety of data sources were checked. The communications between the attacking machines and the affected parties were analyzed

with tcpdump. In some of the tests, it was necessary to look at the logs of the Nexus 1000V to find whether the vulnerability tests had manipulated the switch.

Several tools were necessary to determine whether the Nexus 1000V was vulnerable to each of the vulnerabilities. In some instances, it was necessary to use more than one tool to test for the specific vulnerability. The effects of these potential vulnerabilities were then assessed through several means. A summary of the vulnerabilities and the tools used is depicted in Table 3.3.

Table 3.3.

Summary of Tools Used to Test for Vulnerabilities

Vulnerability	Tool
CAM Overflow	Macof
VLAN Hopping	Nemesis and Yersinia
STP Manipulation	Nemesis and Yersinia
ARP Poisoning	Ettercap
Private VLAN Vulnerabilities	Nemesis

3.4. Distributed Switch Communication Vulnerabilities

This research also examined the communication mechanisms used by the Nexus 1000V to facilitate the VSM to VSM, VSM to VEM, and VEM to VEM communications. With the limited amount of information detailing the communication mechanisms used, it was necessary to first analyze the communication between the distributed switches. This was accomplished by mirroring the port of the physical switch that was being used by one of the ESXi servers so that it would be received by the network-monitoring server. By using tcpdump on the physical monitoring server, it was possible to capture and analyze the traffic being passed between the distributed switches. Traffic was generated by making configuration changes. In particular, virtual machines were moved from one port-group to another through vCenter and changes were also made to the port-profile through the VSM. Specifically, CentOS2 was moved from the

Packet port-group to the SecureVLAN group and later after being placed back in the Packet port-group, the Packet port-group's port-profile was updated to access VLAN 973. The network traffic generated was then assessed; to do this, it was first necessary to determine characteristics of the Nexus's communication traffic to distinguish this network traffic from the other network traffic. Once the identifying characteristics for the distributed switch communication traffic had been determined, the traffic was then analyzed. This analysis consisted of attempting to ascertain the protocols being used and any mechanisms that might be employed to protect the communications. The final step was to attempt to disrupt the communication taking place between the switches. Disruption of the communication was attempted by using the tool tcprelay, which allows captured packets to be replayed (Turner, 2010).

3.5. Virtual Machine Manipulation

The final step of this research was to take a look at the impact an attacker might have were they to create multiple VSM instances. This research began with an analysis of whether it would be possible for an attacker with a compromised ESXi host to add an additional VSM to the network. In particular, it looked at the authentication and authorization mechanisms used by the Nexus 1000V to ensure that that only a legitimate VSM could be added. The second stage of this research involved looking at the repercussions of cloning a VSM. First, a standalone VSM was cloned. The goal of this was to see how vCenter and the VSM handled multiple VSMs. To carry out this test, it was necessary to change the redundancy of the test architecture from high availability to standalone. This was accomplished by powering down the secondary VSM and entering the command, "system redundancy role standalone" at the configuration prompt on the primary VSM. Once the VSM was in standalone mode, the VSM was duplicated through vCenter.

Following the testing of the standalone VSM, the duplicate VSM was deleted and the architecture was reverted to high availability mode. The high availability mode was enabled by powering the secondary VSM back on and issuing the command, "system redundancy role primary" at the configuration prompt on the primary VSM. Once the

Nexus 1000V was back in high availability mode, the effects of cloning the primary VSM were assessed. Following this, the same process was repeated for the duplication of the secondary VSM.

3.6. Summary

In this chapter, the methodology used to evaluate the security implications of the Cisco Nexus 1000V versus traditional physical switches was detailed. It consisted of four steps. The first step was to determine and then create an architecture that accurately reflected the functionality commonly found in implementations of the Nexus 1000V. After the representative architecture had been created, the Nexus 1000V was assessed to find whether it suffered from the common vulnerabilities found on current and previous physical switches. The third step of this research was establishing whether the communication mechanism used between the distributed switches was vulnerable to malicious actions. This research's final step was to assess the adding of VSMs and to identify any negative effects as a result of adding additional VSMs.

CHAPTER 4. RESULTS AND CONCLUSIONS

Throughout the steps of this research, multitudes of results were captured. The purpose of this chapter is to examine the results that were gathered as the steps of the experiments progressed and to draw conclusions from the results. This chapter will begin with a look at the findings following the creation of the test architecture. Following this, the results found while testing for vulnerabilities currently and previously found in physical switches will be examined. The third section of this chapter will cover what was found while examining and analyzing the internal communications used by the Nexus 1000V. After this section, the results of attempting to create additional VSMs will be discussed. Finally, this chapter will conclude with a summary of the covered results and the conclusions that can be drawn from them.

After the creation of the architecture, it was possible to verify the functionality offered by the Nexus 1000V. This was done by utilizing both vCenter's management interface and the Nexus 1000V's remote command line interface, that provides the same method of configuration found in physical switches. It was found that the configuration changes made within the Nexus 1000V was capable of being applied directly through vCenter. For instance, it was possible to create new port-profiles through the command line while connected into the VSM. These port-profiles would then appear as a new port-group in the network adapter dialog used to configure virtual machines. Conversely, the changes such as a change in port-group assignment for a virtual machine, was noticeable in the Nexus's remote access when issuing commands such as "show interface status" and "show port-profiles". Furthermore, it was found that the virtual machines that had been connected to the Nexus 1000V, had proper network connectivity and were able to communicate with one another as well as outside the virtual environment.

The creation of the architecture also provided a better understanding of the inner-workings of the Nexus 1000V. In particular, it was found that the Nexus 1000V required a minimum of three VLANs. These VLANs were the Control, Packet, and Management VLANs. The control VLAN was found to be used to facilitate VSM to VSM and VSM to VEM communications. Later in this chapter, the communications mechanisms used will be discussed in further detail. It was found that the packet VLAN served as the VLAN that was meant to provide network connectivity. It should be noted that network connectivity was not limited to this VLAN. Configuration changes on the Nexus 1000V allowed additional VLAN to be used and were assigned to the port-groups described in the methodology section. Finally, the management VLAN was used by the Nexus 1000V to provide remote access. In particular, it allowed the switch to be accessed via Secure Shell (SSH).

4.1. Physical Switch Vulnerabilities

The first tests of this research took a look at the vulnerabilities that presently or had previously been found on physical switches and sought to determine whether they affected the Nexus 1000V. To make this determination, a variety of tests were carried out. In the remainder of this section, the results of the tests for physical switch vulnerabilities will be detailed. It will conclude with a summary of the findings and conclusions that can be made from the results.

4.1.1. CAM Overflows

Using macof it was possible to test the switch's vulnerability to CAM overflows. The functionality of macof was verified by capturing packets on the virtual monitoring servers residing on each of the ESXi hosts. These virtual monitoring servers were also connected to VLAN 972. It is important to note that the VSM had no port or VLAN mirroring configured; instead, the monitoring servers only received the same traffic the other virtual machines on the VLAN received. Figure 4.1, shows a subset of the packets generated by macof. The reason for only providing a subset is that presenting all of the

packets would have required excessive space and would have provided little to no additional value.

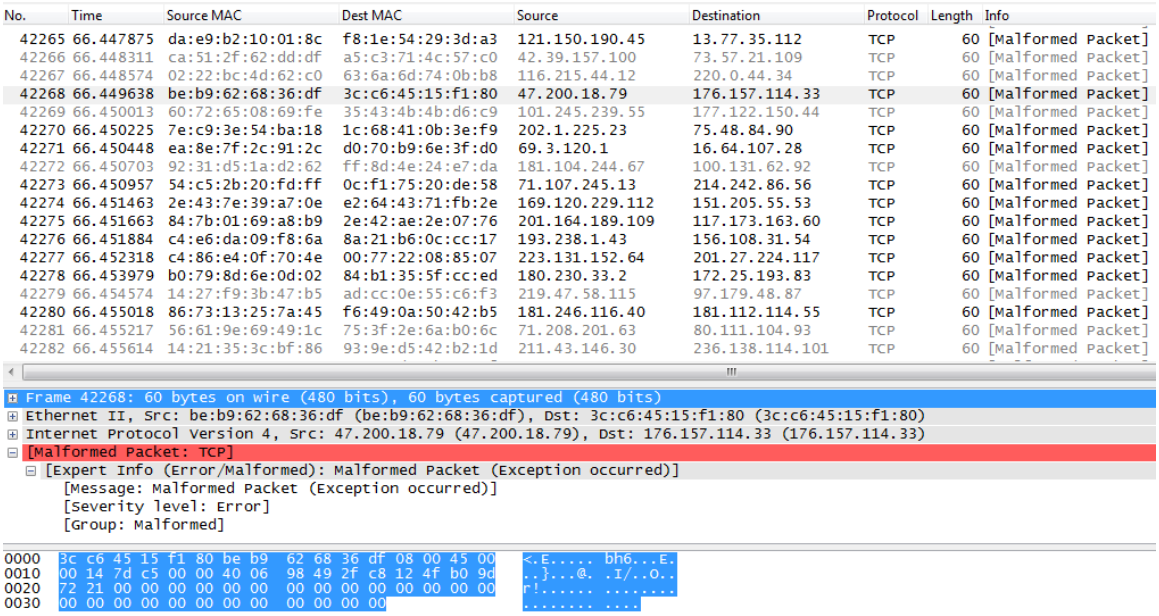

Figure 4.1. CAM overflow packets generated by macof.

As the packets were being generated by macof, the impact that it was having on the switch was verified by checking the number of MAC addresses the Nexus 1000V was tracking. This was achieved by issuing the command “show mac address-table count vlan 972”. It was found that the Nexus was tracking 12,288 MAC addresses for the VLAN 972. After the attack was completed, the packets captured by the three virtual monitoring servers were assessed. It was found that two of the virtual machines had received ICMP packets that had been meant for one of the legitimate CentOS hosts. In each case, the traffic had originated from a virtual machine that existed on the same ESXi host as the virtual monitoring server. The observed ICMP packets are depicted in Figures 4.2 and 4.3.

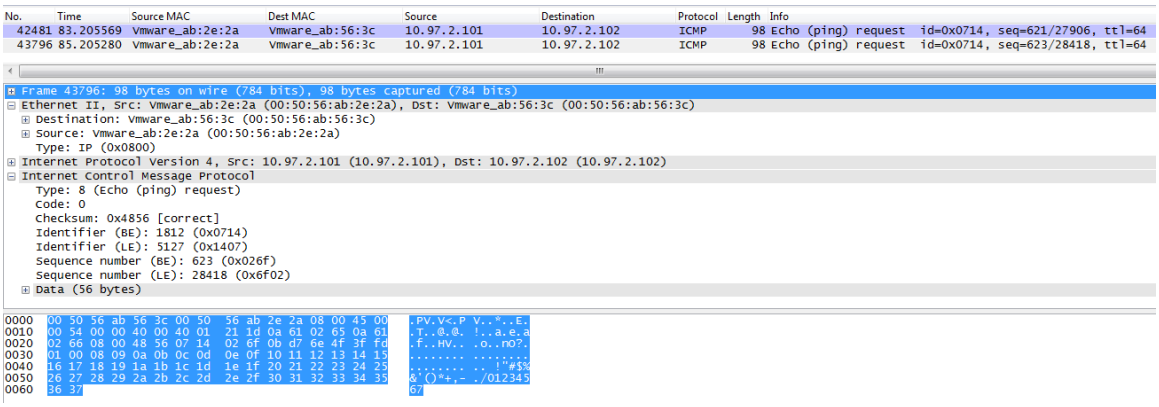

Figure 4.2. ICMP packets captured on vSniff1.

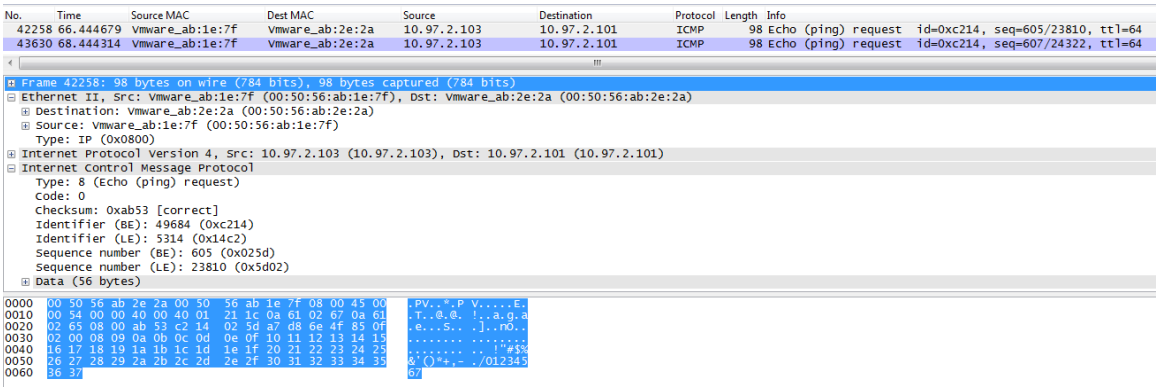

Figure 4.3. ICMP packets captured on vSniff3.

4.1.2. VLAN Hopping

The use of Yersinia allowed for the testing of two techniques of VLAN hopping. Before the tests could be carried out, it was necessary to ensure that there was indeed an access list preventing communication from the 974 VLAN to the 972 VLAN. This was accomplished by monitoring the attack virtual machine's console and configuring the VSM to replicate all traffic entering and leaving the attack machine that was residing on the 974 VLAN. An attempt to ping the CentOS2 virtual machine residing on the 972 VLAN was then made. Through monitoring the console of the attack host, it was found that traffic was being blocked, as the response received was "From 10.97.4.1 icmp_seq=1 Packet filtered", with the sequence number incrementing with each ping attempt. This was then verified by the packets captured on the virtual monitoring server that had

received the replicated network traffic. A subset of the packets captured is depicted in Figure 4.4.

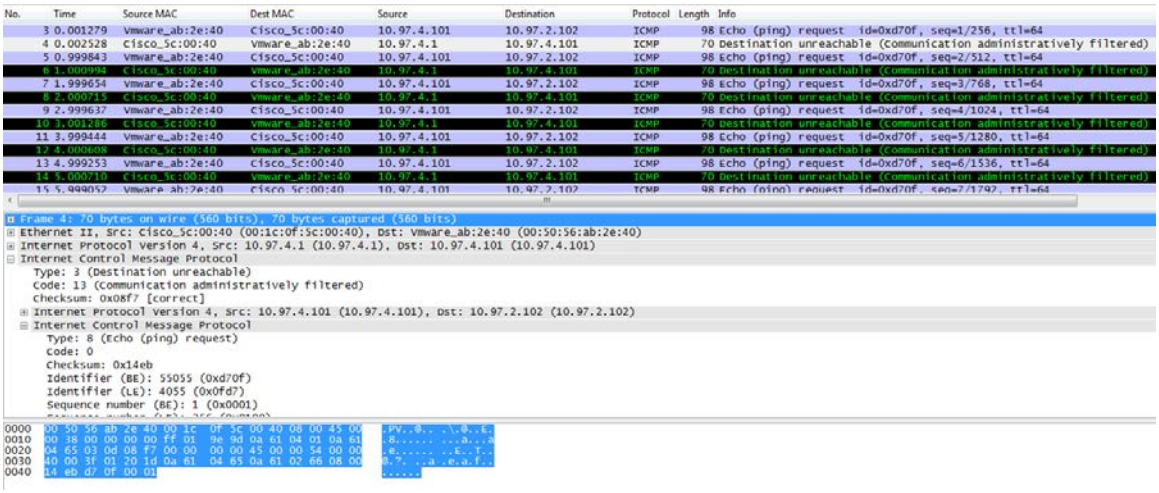

Figure 4.4. Packet capture verifying packet filtering was taking place.

Upon verification that traffic between VLAN 974 and 972 had been restricted, the first method of VLAN hopping was attempted. To verify that Yersinia was correctly creating packets with the destination MAC address of the CentOS 2 virtual machine, the monitoring setup used to verify the VLAN restrictions was again used. By monitoring VLAN 974, it was possible to analyze the packets being sent by Yersinia. It was found that Yersinia was correctly creating manipulated packets. Figure 4.5 shows a collection of the captured packets and the manipulated destination MAC address.

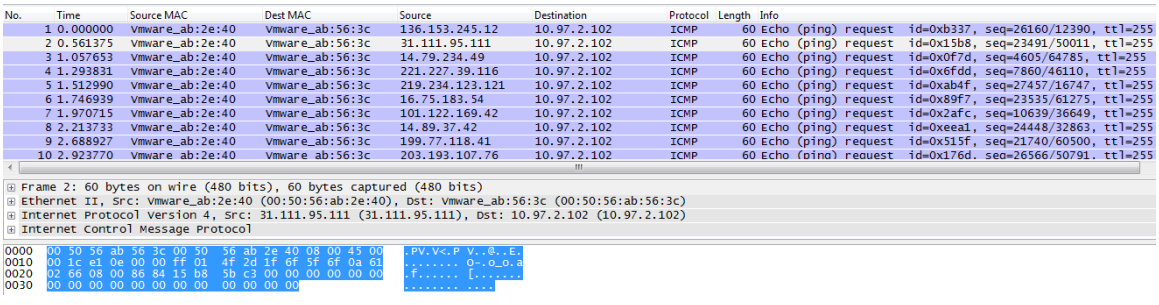

Figure 4.5. Packets generated by Yersinia with manipulated MAC addresses.

With the manipulation of the packet verified, whether the packet reached the virtual machine needed to be ascertained. This was accomplished by again making use of the virtual monitoring server; however, this time the VSM was configured to replicate the traffic destined for the CentOS2 virtual machine to the vSniff2 virtual monitoring server. This setup allowed the monitoring server to capture all of the network traffic going to and from the CentOS2 virtual machine. It was found that the Nexus 1000V was not ignoring the VLANs and forwarding the packets, as none of the packets that were created by the attack machine reached the CentOS2 virtual machine.

The second VLAN hopping test involved the creation of packets with two 802.1Q headers. Like several of the other experiments, Yersinia was used to create these packets. Before the Nexus 1000V's handling of packets with two 802.1Q headers could be assessed, it was necessary to verify that Yersinia was properly creating these packets. This was achieved by running packet capture software on the CentOS attack virtual machine. With this, it was found that Yersinia was indeed creating packets with two 802.1Q headers. Figure 4.6 shows one of the captured packets that contain two 802.1Q headers.

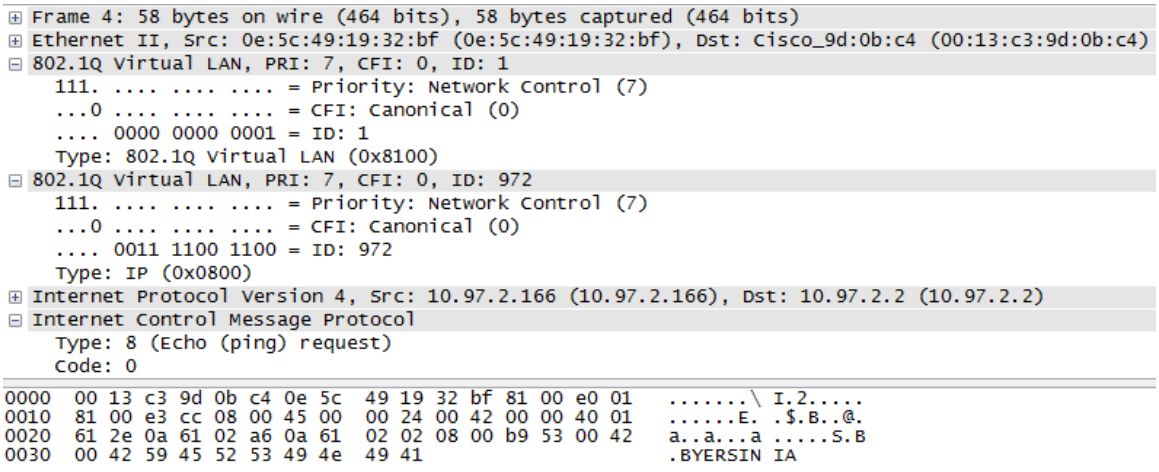

Figure 4.6. Packets generated by Yersinia with two 802.1Q headers.

It was also necessary to ensure that the Nexus 1000V was removing the initial 802.1Q header while keeping the second header. This was achieved by configuring the

VSM to replicate the traffic on VLAN 1, the native VLAN, and to send this traffic to the virtual monitoring server vSniff 1. Figure 4.7 shows one of the captured packets, with its initial 802.1Q header removed.

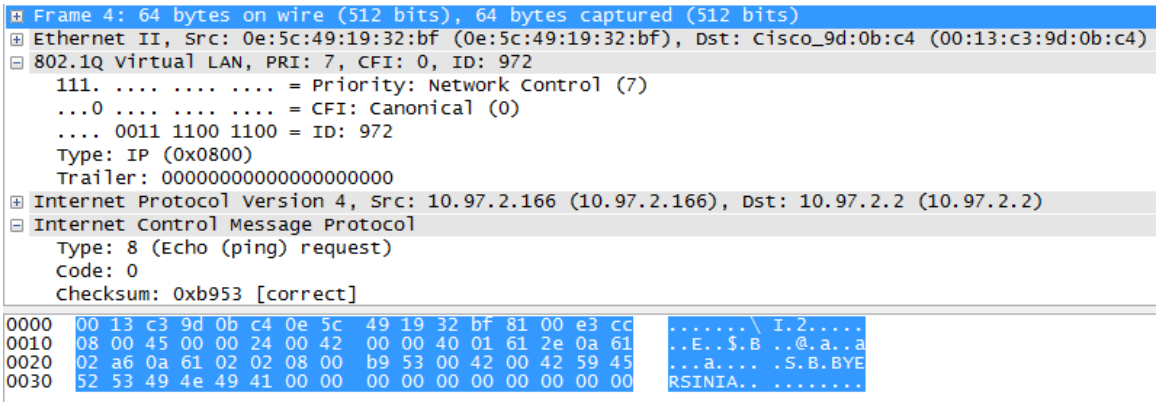

Figure 4.7. Packets generated by Yersinia with one 802.1Q header removed that were received on the Nexus 1000V.

The last step in looking at this type of VLAN hopping was to determine if this packet was being sent to the physical switch. By configuring the physical switch to replicate the traffic on VLAN 1 and 972 and having it sent to the physical monitoring server, Sniffer, monitoring of the traffic was possible. It was found that the crafted packet was indeed being processed and forwarded by the Nexus to the physical switch. Interestingly, the packet stayed on the native VLAN and did not “hop” over to VLAN 972. Nonetheless, this observation meant that it is possible for a virtual machine to create a packet that could “hop” into another VLAN after it has been passed to another physical switch. Figure 4.8 depicts one of the crafted packets that was captured on the Sniffer physical monitoring server.

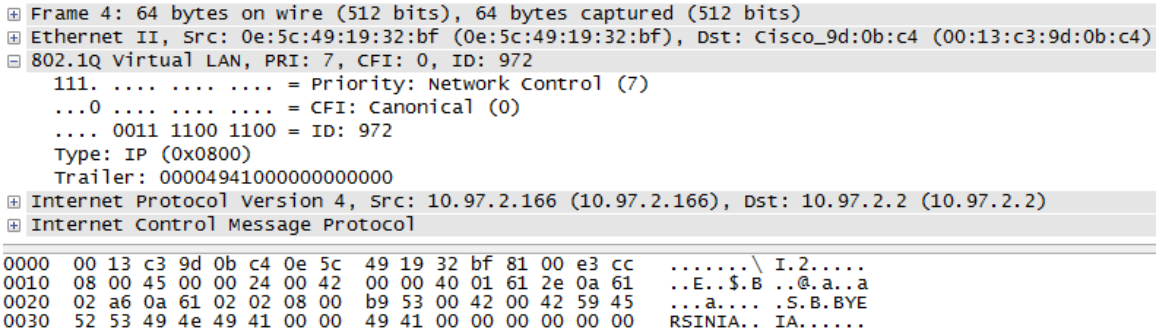

Figure 4.8. Packets generated by Yersinia with one 802.1Q header removed received on the physical switch.

4.1.3. STP Manipulation

CentOS2 was used again, this time to test whether the Nexus 1000V was susceptible to STP manipulation. In the tests, two different STP manipulations were attempted, with both attempting a different form of denial-of-service attack. For each of these attacks it was necessary to verify that Yersinia was generating the appropriate packets. This was accomplished using a monitoring virtual machine and having the VSM replicate all of the attack virtual machine's traffic to the virtual monitoring server. A subset of the packets captured for verification of the Configuration BPDU denial-of-service and the Topology Change Notification BPDU denial-of-service are depicted in Figures 4.9 and 4.10 respectively.

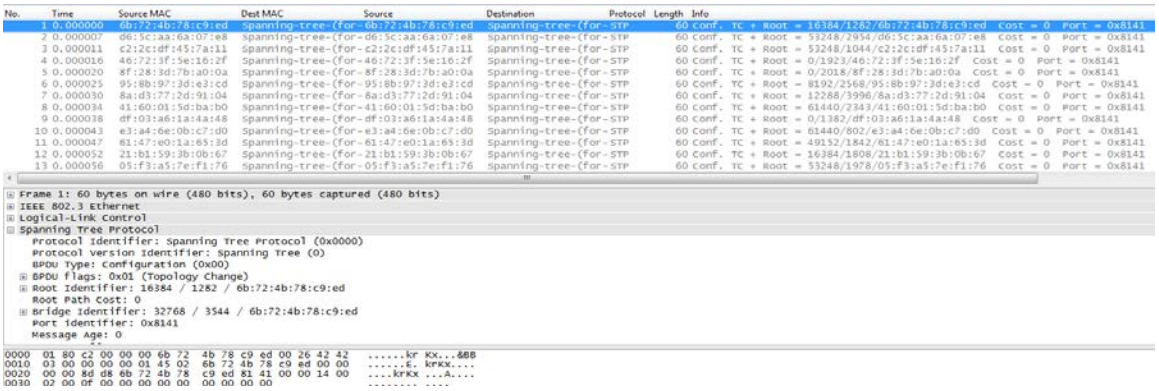

Figure 4.9. STP Configuration BPDUs generated during the denial-of-service attempt.

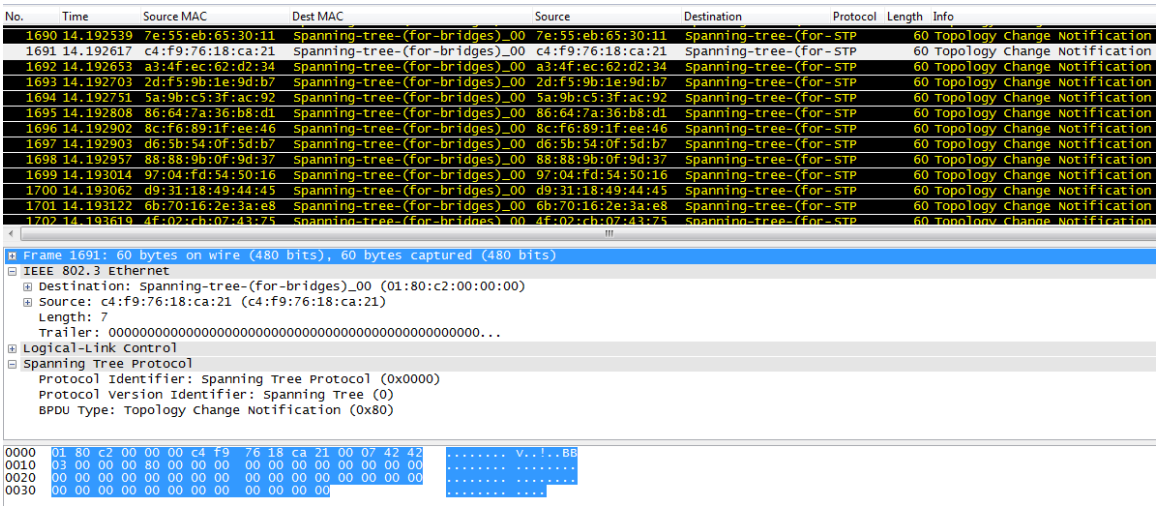

Figure 4.10. STP Topology Change Notification BPDUs generated during the denial-of-service attempt.

After the packets being sent by Yersinia had been verified, their impact on the Nexus 1000V was assessed. The most apparent observation was that the switch's functionality had persisted despite the attempted manipulations. Had the attack been successful, the switch would have been overwhelmed and its functionality would have been impacted. If the attempted attacks were successful, it would cause the switch to run processor intensive computations Because of this, the CPU usage of the VSM in vCenter was checked to further verify the impact of the attempts. It was found that the CPU usage had remained at a level consistent with normal usage. The graph depicted in Figure 4.11 shows the CPU usage of the VSM virtual machine as noted by vCenter. The Configuration BPDU denial-of-service attempt ran from 4:10 P.M. to 4:20 P.M. and the Topology Change Notification ran from 4:30 P.M. to 4:40 P.M. For both attempts, there was little to no deviation from the normal CPU usage on the VSM.

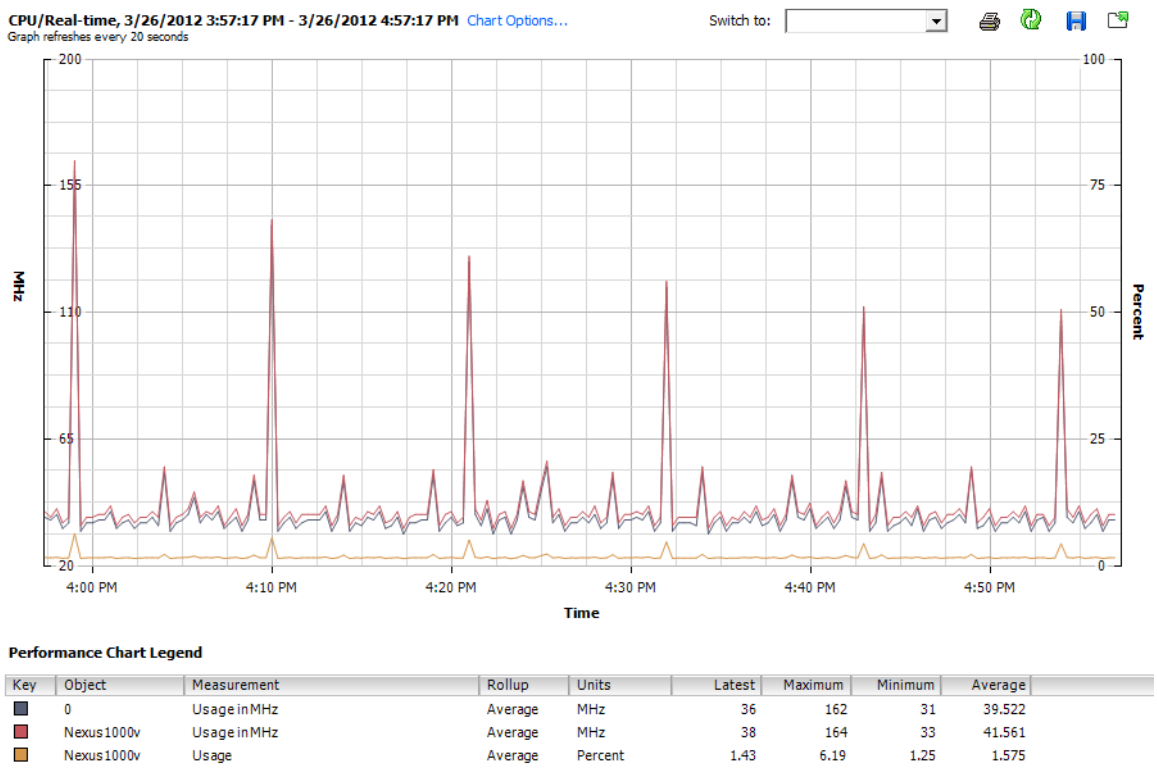

Figure 4.11. The CPU usage on the primary VSM during the STP denial-of-service attempts.

To determine whether the manipulations's impact had been limted to the VEM residing on the host on which the attacker existed, the CPU usage of the ESXi host was examined. As depicted in the CPU usage graph in Figure 4.12, it was found that during the attacks, the CPU usage on the ESXi host spiked dramatically during the attempted manipulations. When compared with the CPU usage of the attack virtual machine, it was found that the increase in CPU usage was likely due to the usage by the attacker virtual machine and the switch handling the excessive number of packets being recieved. The CPU usage of the attacking virtual machine during both of the attacks is depicted in Figure 4.13.

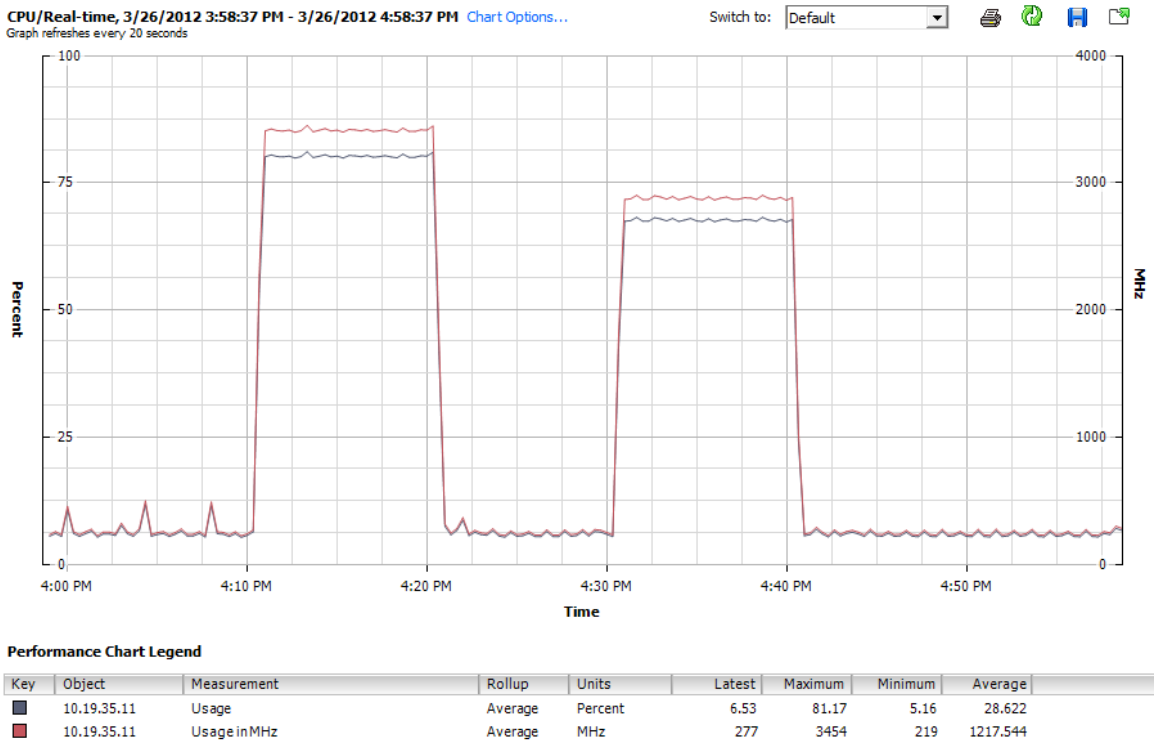

Figure 4.12. The CPU usage on the attacker's ESXi host during the STP denial-of-service attempts.

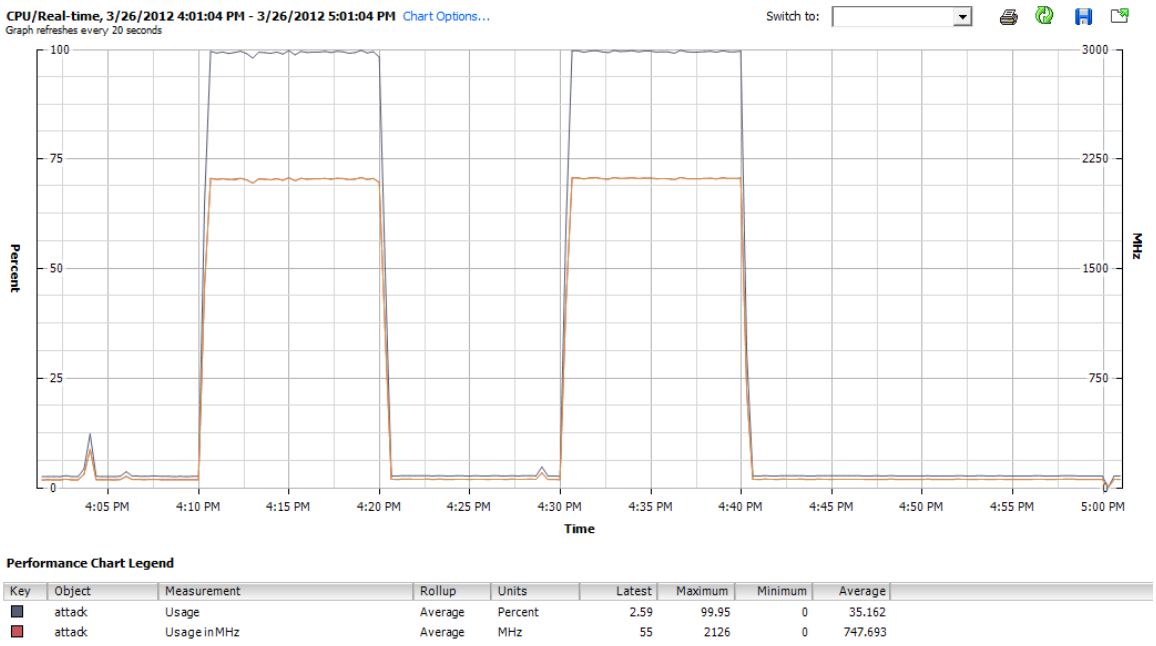

Figure 4.13. The CPU usage on the attacker's virtual machine during the STP denial-of-service attempts.

The second portion of the STP tests involved testing whether the Nexus 1000V would be affected by Cisco's proprietary version of STP, PVST+. For this test, it was necessary to use Nemesis along with a script that would automatically generate random MAC addresses. Using Nemesis and the created script, it was possible to attempt Configuration and Topology Change Notification BPDU denial-of-service tests. Packets from these tests were captured in the same manner as the previous STP tests. Once captured, these packets were verified to ensure that the tests were being properly carried out. Figures 4.14 and 4.15 respectively depict a subset of the packets captured from the Configuration BPDU and Topology Change Notification BPDU denial-of-service attempts.

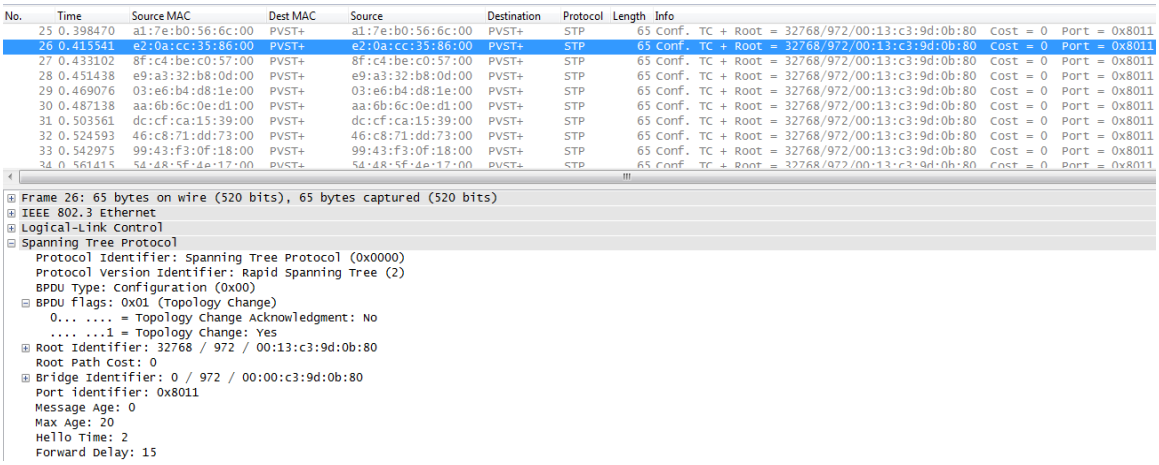

Figure 4.14. PVST+ Configuration BPDUs generated during the denial-of-service attempt.

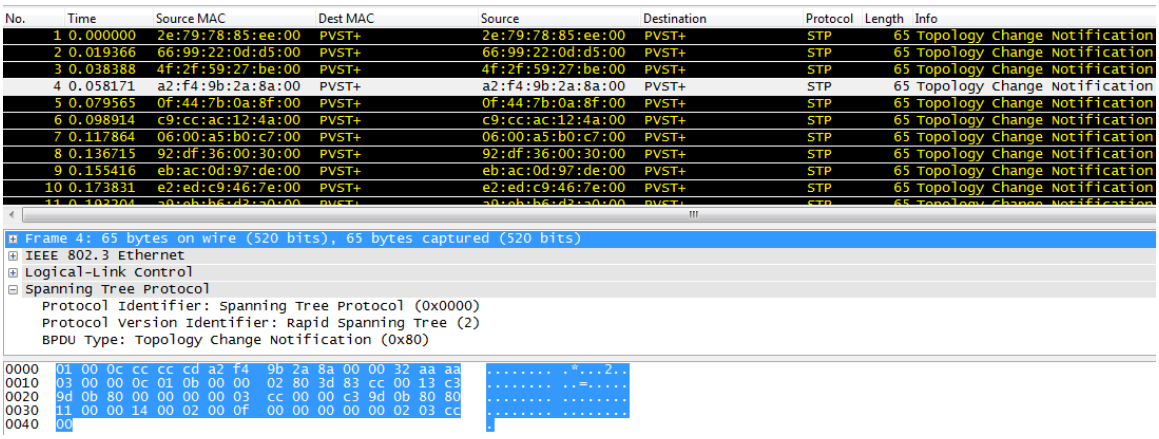

Figure 4.15. PVST+ Topology Change Notification BPDUs generated during the denial-of-service attempt.

As with the first set of STP tests, it was found that the PVST+ tests did not impact the network functionality provided to the virtual machines. Like in the previous tests, the CPU usage of the VSM was monitored to look for clues that it might be processing the packets and recalculating the STP topology. Figure 4.16 depicts the CPU usage of the VSM during the two attacks. It should be noted that the Configuration BPDU test began 5:30 PM and ended at 5:40 PM. The Topology Change Notification BPDU test began at 5:50 PM and ended at 6:00 PM.

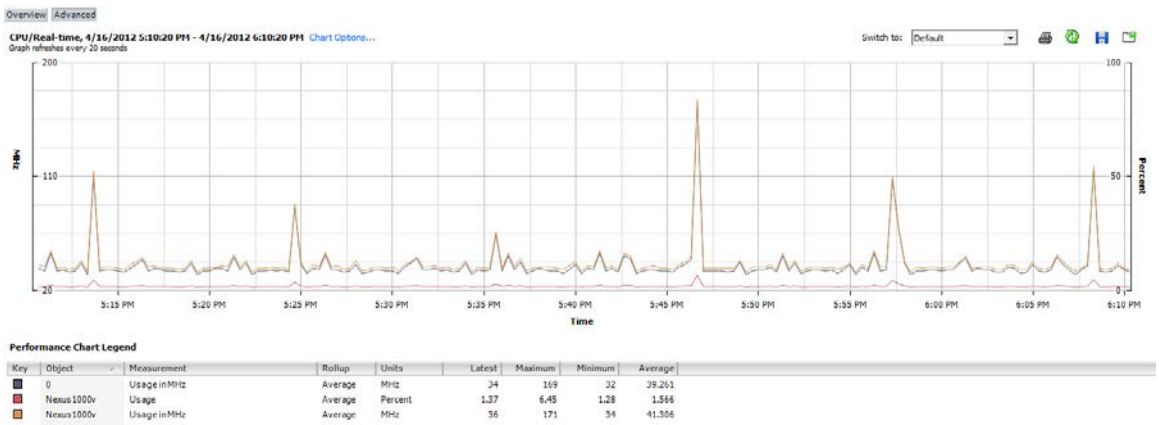

Figure 4.16. The CPU usage on the primary VSM during the PVST+ denial-of-service attempts.

Just as in the initial STP tests, the CPU usage of the ESXi host that the attacker resided on was examined in an attempt to verify whether the VEM had been affected. It was found that during the attacks, the CPU usage on the ESXi host spiked dramatically during the attempted manipulations. These results are depicted in Figure 4.17. When compared with the CPU usage of the attack virtual machine, it was found that the increase in CPU was likely due to the processor usage by the attacker virtual machine and the switch handling the received packets. The CPU usage of the attacking virtual machine during these attacks is depicted in Figure 4.18.

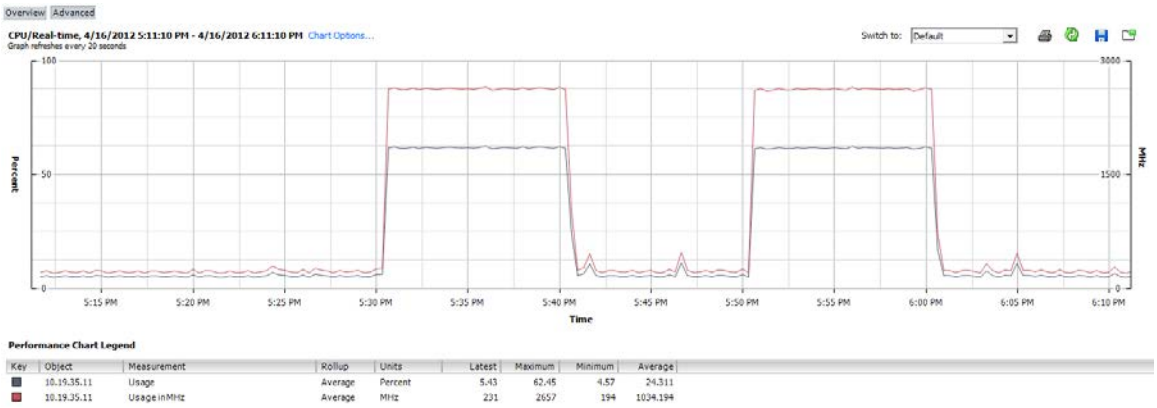

Figure 4.17. The CPU usage on the attacker's ESXi host during the PVST+ denial-of-service attempts.

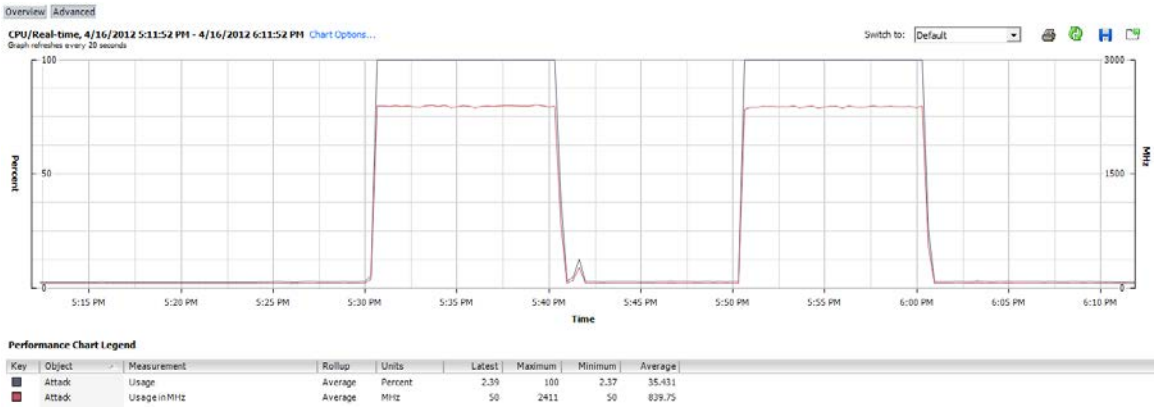

Figure 4.18. The CPU usage on the attacker's virtual machine during the PVST+ denial-of-service attempts.

4.1.4. ARP Poisoning

The tool, Ettercap allowed for the Nexus 1000V's vulnerability to ARP poisoning to be assessed. After the ARP poisoning had been carried out, it was possible to verify its effectiveness by capturing the packets being received on the attacking virtual machine. It was found that the Nexus 1000V was susceptible to ARP poisoning as traffic that was meant for other virtual machines was being received by the attacking machine. Figure 4.19 depicts a capture of some of the packets created by Ettercap and Figure 4.20 depicts a subset of the captured ICMP packets that were supposed to be amongst the legitimate CentOS hosts residing on the VLAN.

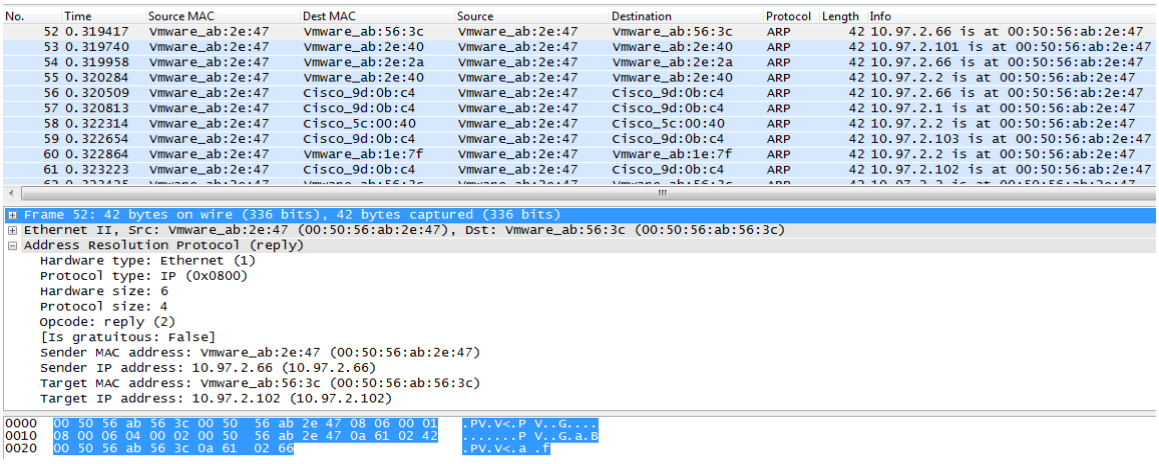

Figure 4.19. ARP traffic generated by Ettercap.

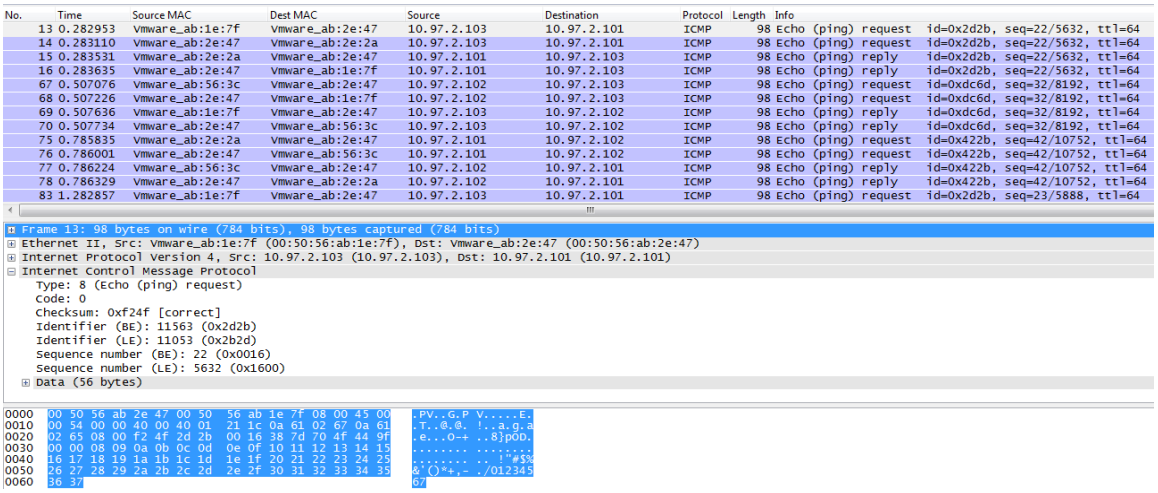

Figure 4.20. ICMP packets that were captured during the ARP poisoning test.

4.1.5. Private VLAN Vulnerabilities

The final physical vulnerability that was tested for was private VLAN vulnerabilities. Prior to testing for this vulnerability, it was necessary to ensure that the private VLAN in isolation mode was indeed preventing the hosts from communicating with one another. This was accomplished by having the VSM mirror the private VLAN so that the packets on the private VLAN could be monitored by the virtual monitoring server. While the packets were being captured, CentOS2 and CentOS3 attempted to ping each other. These attempts failed and the machines reported that the destination was unreachable. In Figure 4.21, it can be seen that while the virtual machines attempted to ping each other, they needed to learn the MAC addresses of each other but were unable to learn the necessary information.

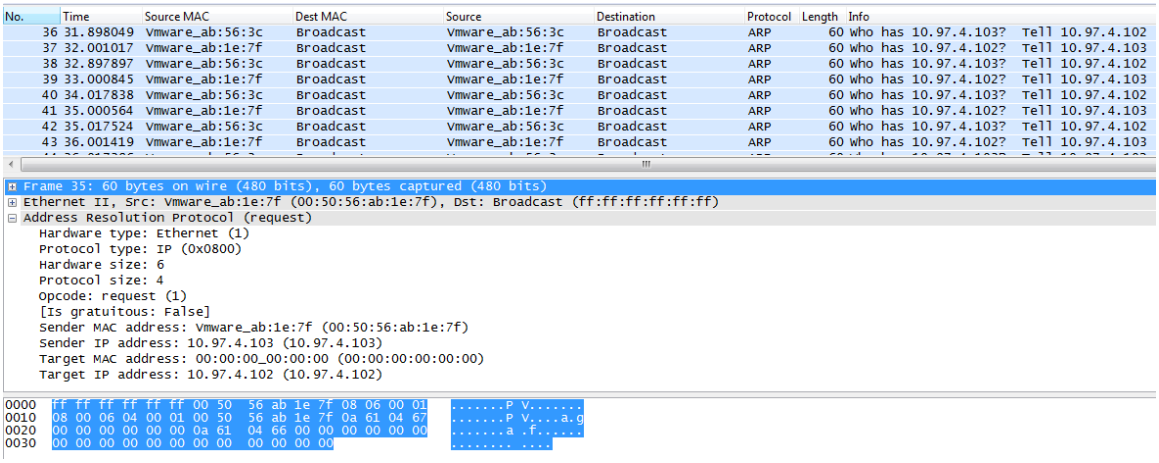

Figure 4.21. A packet capture of the virtual machines on the private VLAN attempting to ping each other.

To verify that the virtual machines still had network connectivity, both virtual machines attempted to ping their default gateway. Both of the virtual machines were able to ping their gateway successfully. Figure 4.22 shows both of the virtual machines being able to ping their gateway successfully.

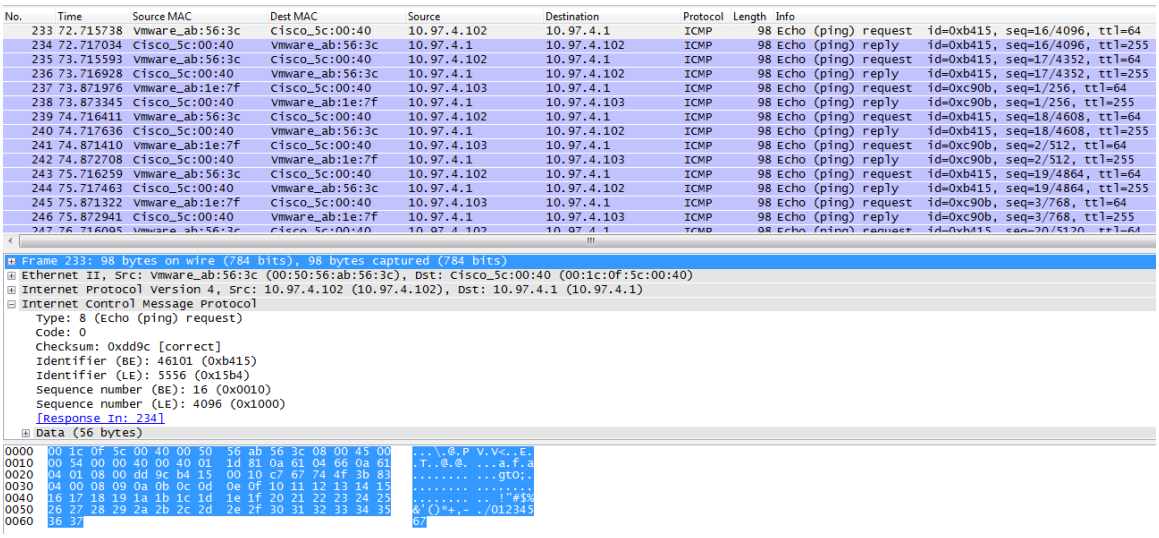

Figure 4.22. A packet capture of the virtual machines on the private VLAN pinging their gateway.

With the functionality of the private VLAN verified, it was then possible to assess the Nexus 1000V's susceptibility to private VLAN attacks. As stated in the methodology section, this test was carried out by using Nemesis. Through the use of Nemesis, it was found that like physical switches, the Nexus 1000V was vulnerable to private VLAN attacks. This determination was made because the crafted packet from the Backtrack virtual machine was able to reach CentOS2, despite both machines being on the private VLAN. It should be noted that this communication was unidirectional, as the CentOS2 virtual machine was still unable to ascertain the MAC address of the Backtrack virtual machine and therefore was unable to respond. Figure 4.23 depicts a packet capture where the Backtrack virtual machine was able to transmit ICMP traffic to the CentOS2 virtual machine.

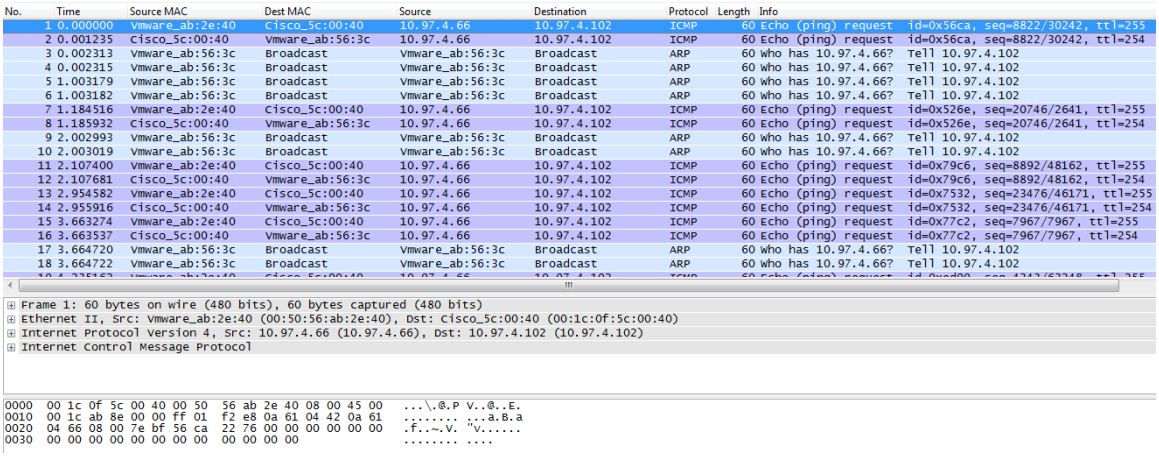

Figure 4.23. Packet capture of the attacking Backtrack virtual machine successfully sending ICMP traffic to the CentOS2 host.

4.1.6. Physical Switch Vulnerabilities Summary and Conclusions

Upon completion of the tests for vulnerabilities that currently or have previously affected physical switches, a better understanding of the Nexus 1000V had been gained. In particular, where it stood against physical switches in terms of security was better understood. It was found that some of the vulnerabilities that affect physical switches are also present in the Nexus 1000V. Of those tested for, it was found that the Nexus 1000V is vulnerable to CAM overflows, 802.1Q double-header VLAN hopping, ARP poisoning, and private VLAN vulnerabilities. Although these vulnerabilities exist on the Nexus 1000V and potentially threaten the security of the switch, there are methods for mitigating their risk. Indeed, many measures that have been used to protect physical switches are available on the Nexus 1000V. For instance, it is recommend to use Cisco's port security features, which are available on the Nexus 1000V, to prevent both ARP poisoning and CAM overflows (Bastien, et al., 2006). To prevent 802.1Q double-header VLAN hopping, it is suggested that the native VLAN not be used for anything other than switch-to-switch communications. In other words, if the Nexus 1000V is properly configured and the native VLAN is not accessible to the virtual machines, this issue can be mitigated. Finally, private VLAN vulnerabilities can be eliminated if the router that is

used as the gateway is configured with an access-list that prevents the devices on the private VLAN from communicating with one another.

STP manipulation and the VLAN hopping technique that used manual MAC addresses were found to not affect the Nexus 1000V. Both of these results were somewhat logical. In the product literature provided by Cisco, it was stated that the Nexus 1000V no longer used STP to eliminate network loops (Cisco, 2011b). Despite this, the tests were carried out to ensure that there was no lingering aspect of the STP that could affect the security of the Nexus 1000V. In the end, it was found that there were no apparent remnants of STP or PVST+ causing security issues. The absence of VLAN hopping using manual MAC addresses was also logical since the preventative VLAN techniques that Farrow described in high-end switches in 2003, are now commonplace (Farrow, 2003).

Although some of the vulnerabilities found on physical switches have been found to affect the Nexus 1000V, if proper configuration considerations are made these issues can be mitigated. Furthermore, it was found that unlike most physical switches, the Nexus 1000V does not use a form of STP and is therefore not vulnerable to its manipulation. Because of the possibilities for mitigation in the found vulnerabilities and the absence of STP manipulation, it was determined that within the context of the tested vulnerabilities, the security implications of the Nexus 1000V were less than those presented by using physical switches.

4.2. Distributed Switch Communications

Through the use of packet captures, it was possible to gain a better insight into the inner-workings of the Cisco Nexus 1000V. These packets were captured in hopes of overcoming the limited amount of information available about how the VSMs and VEMs communicated amongst each other. Furthermore, they allowed for a better understanding of potential security concerns that might exist.

4.2.1. Initial Analysis

To capture the packets, it was first necessary to understand where the VSM and VEM communications were taking place. In the installation and configuration guide, it was noted that the Control VLAN was used to facilitate communications amongst the virtual switches (Cisco, 2012). Because of this, the control VLAN was examined. What was thought to be communications between the VSMs and VEMs was found. Since the captured communications were using a data link protocol, the MAC addresses of the communications and the MAC addresses stored in the MAC address table of the Nexus 1000V were then compared. This analysis of the MAC addresses made it possible to identify the switches that were communicating with each other. It was found that there were communications being sent amongst the VSMs and VEMs. After analyzing the packets, it was also found that the VEMs did not appear to communicate with each other. Instead, the captured VEM communications were only with the VSM. During normal conditions, the primary VSM would send out a broadcast message. After receiving the message, both of the VEMs responded to the primary VSM, which then followed up with another response packet. These communications repeated every second. It is important to note that that the secondary VSM was not involved with any communications with the VEMs and did not respond to the broadcast message sent by the primary VSM. Figure 4.24 depicts a subset of the captured communications between the primary VSM and the VEMs.

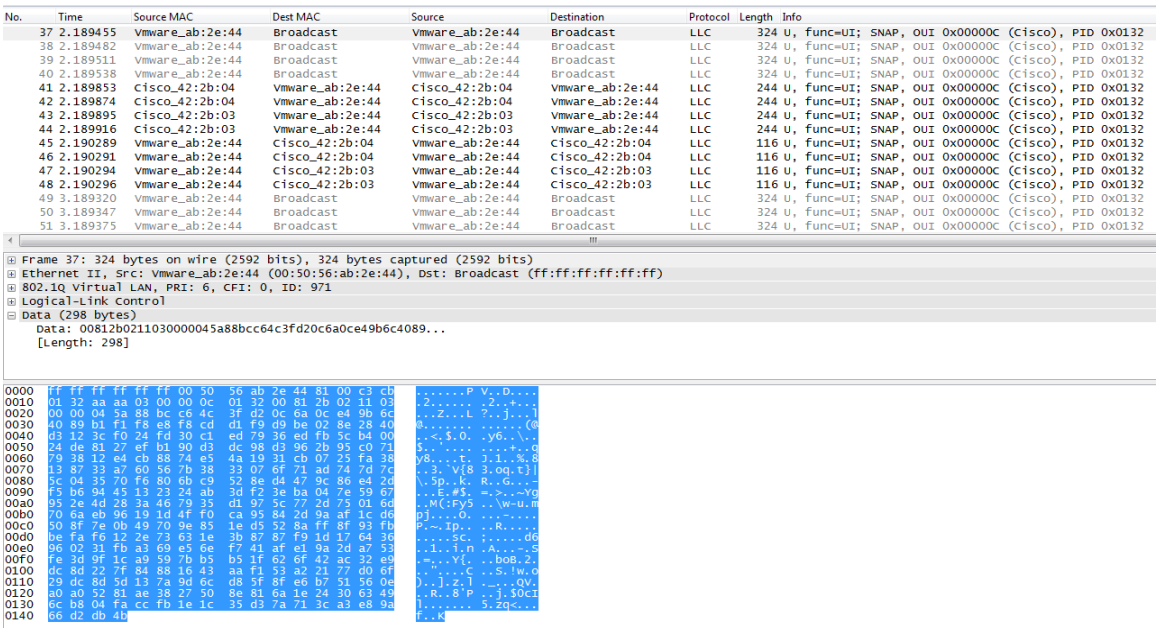

Figure 4.24. Packet capture of normal VSM to VEM communications.

Communications between the primary and secondary VSM were also captured. Figure 4.25 shows a subset of packets captured between the primary VSM and the secondary VSM. In this packet capture, it can be seen that the VSMs are in near constant communications with one another. It appeared that these communications would begin when the primary VSM would send a packet with a length of 276 bytes. After the initial packet was sent by the VSM, between seven and twelve packets were sent amongst the primary and secondary VSMs. These exchanges were found to repeat every second.

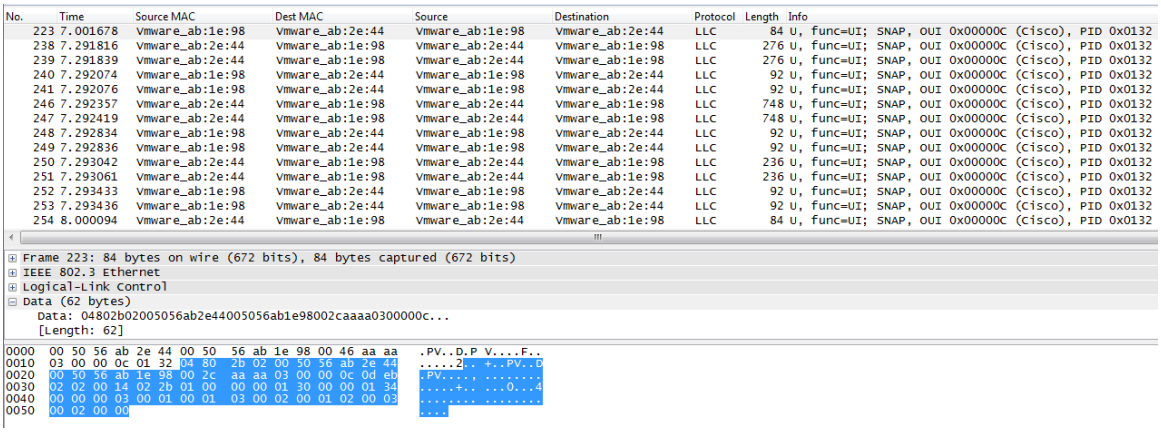

Figure 4.25. Packet capture of the normal VSM to VSM communications.

4.2.2. Analysis of Configuration Communications

Initially only the traffic during normal conditions was examined. To gain a better understanding of how configuration changes were sent between the primary VSM and VEMs, a virtual machine's port-group was changed. The packets sent between the VSMs and the VEMs were then captured. After analyzing the packet capture, it was found that there were two bursts of traffic between the primary VSM and the VEMs. The first burst of traffic began right after the port-group assignment had been changed and the second burst of traffic took place just more than 5 seconds after the initial burst had begun. It was found that this burst in communications was between the primary VSM and the affected VEM. No additional communications took place between the primary VSM and the unaffected VEM. Figure 4.26 depicts the packets per second sent between the primary VSM and the VEMs. It was also found that there were bursts in network traffic between the primary and secondary VSM during the same period of increased traffic between the primary VSM and the affected VEM. Figure 4.27 depicts the packets per second sent between the primary and secondary VSM.

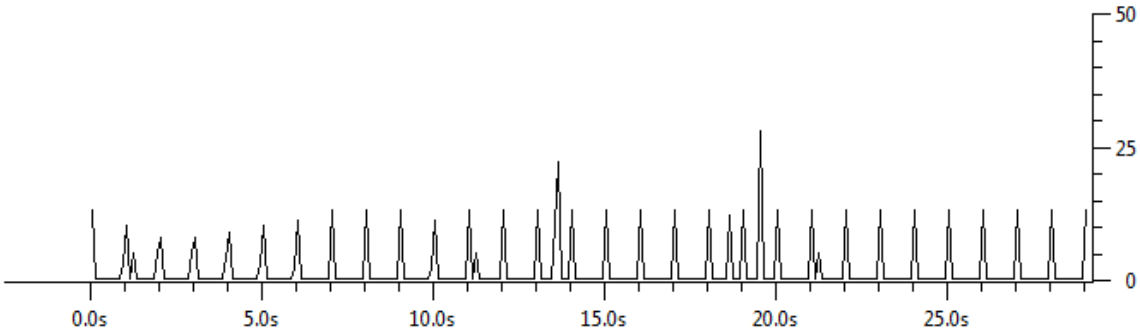

Figure 4.26. Packets sent per second between the primary VSM and VEMs after a virtual machine's port-profile assignment had been changed.

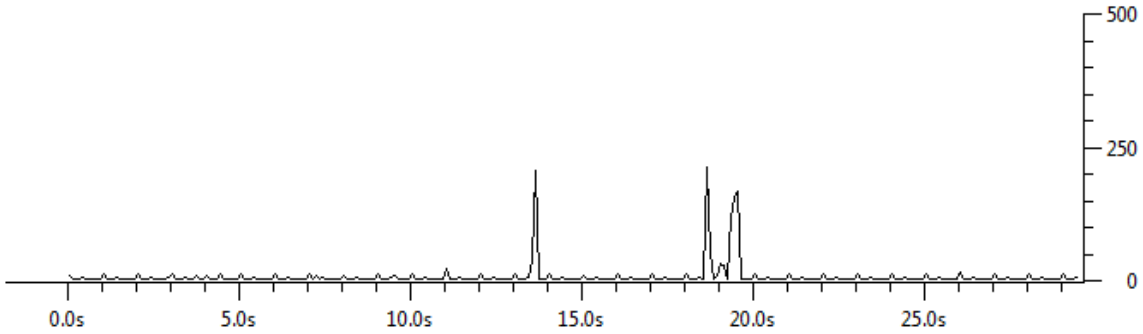

Figure 4.27. Packets sent per second between the primary and secondary VSM after a virtual machine's port-profile assignment had been changed.

Once the port-group change initiated from vCenter had been analyzed, the effect of changing a port-profile configuration through the VSM was assessed. Using the captured packets, it was possible to once again analyze the traffic patterns generated by the changes. Unlike the previous test, it was found that there were three bursts in traffic between the primary VSM and the affected VEM. As with the previous test, no additional communications were sent between the VSM and the unaffected VEM. Figure 4.28 depicts the packets per second sent between the primary VSM and the VEMs. After analyzing the primary VSM to VEM traffic, the traffic between the primary VSM and the secondary VSM was analyzed. It was found that there were two spikes in communications that coincided with the first and third spike of the primary VSM to VEM communications. Figure 4.29 depicts the packets per second sent between the primary and secondary VSM.

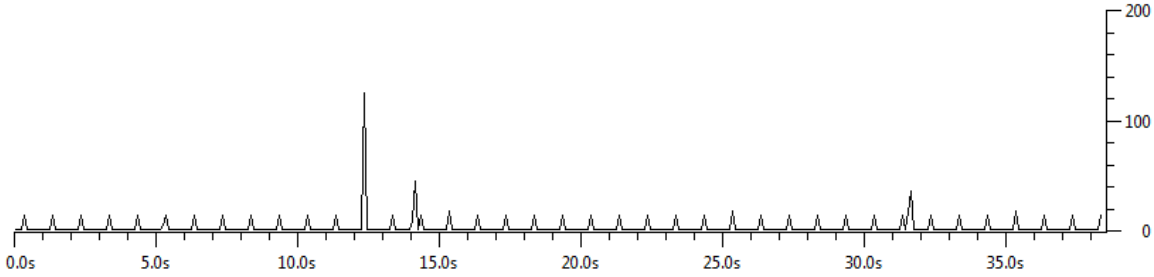

Figure 4.28. Packets sent per millisecond between the VSM and VEMs after a port-profile's configuration had been changed.

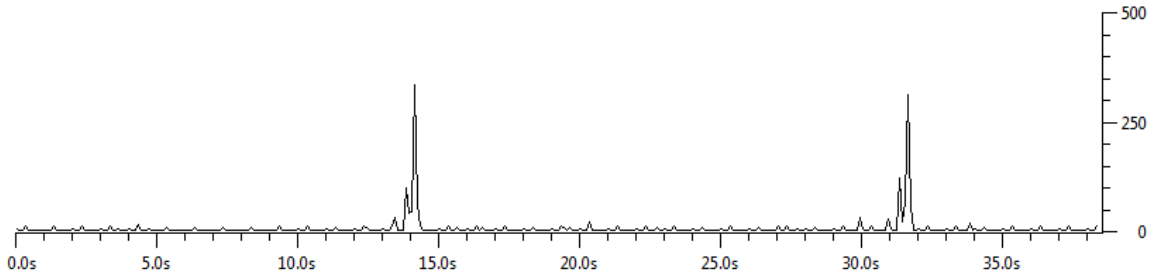

Figure 4.29. Packets sent per millisecond between the VSM and VEMs after a port-profile's configuration had been changed.

After analyzing the content of the packets being sent between the VSMs and the VEMs, it was found that the communications were taking place at the data link layer. The data link layer protocol was not capable of being decoded by Wireshark. In an attempt to learn more information about the protocol being sent by the VSMs, the SNAP header in the packet was examined. Within the SNAP header, it was found that the protocol's organization code was 0x00000C, which Wireshark noted as being Cisco's organizational code. It was also found that the protocol ID was 0x0132. Research into this protocol ID proved inconclusive. With this, it was assumed that the protocol was a proprietary one that had been developed by Cisco. This was validated when a post from a Cisco employee was found on a Cisco support message board stated that the protocol used on the Control VLAN was AIPC, a Cisco proprietary protocol (Mipetrin, 2010). Also in the post, the employee stated that the protocol was encrypted.

Although one might expect the same protocol to be used to for communication between the primary and secondary VSM, this turned out to not be necessarily true. Initially it was thought the same protocol was being used due to the VSM to VSM traffic being a data link protocol and with it having the same identifiers in its SNAP header. Unlike the VEM traffic, these communications did not appear to be encrypted. The most obvious characteristic that led to this determination was the VSM configuration being sent in clear text from the primary VSM to the secondary VSM. One of the captured packets that show the configuration in clear text is displayed in Figure 4.30.

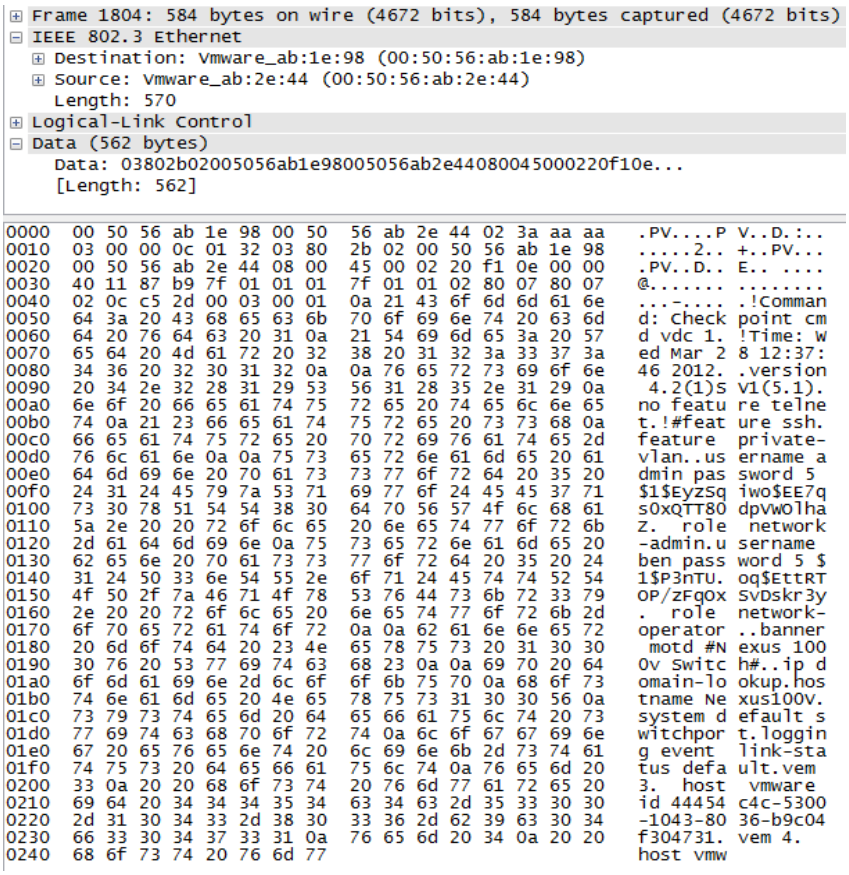

Figure 4.30. A captured packet containing clear text sent from the primary VSM to the secondary VSM.

4.2.3. Analysis of Clear text Communications

Since clear text communications were found, these communications were explored further in search of manipulating them. Before these communications could be manipulated, it was necessary to better understand how the protocol was structured. This was accomplished by reviewing the packets and identifying characteristics and patterns of the communications.

One of the most apparent characteristics identified was that the packets containing clear text configuration information sent by the primary VSM all had a length of 584 bytes. The only exception to this was the packet containing the final portion of the switch's configuration. This packet's length appeared to be dependent on the amount of the configuration that needed to be sent. It should be noted that of all of the other

communications sent by the VSM, only packets containing clear text had a length of 584 bytes. Next, it was found that following each clear text configuration communication sent by the primary VSM, the secondary VSM appeared to acknowledge the primary VSM's communications. This was evident in the packets with a length of 72 bytes that followed each of the primary VSM's packets.

It was also found that although the primary VSM's traffic contained clear text configuration information, there was some non-clear text data in the packet's payload. In particular, this data appeared in the 50 bytes of the payload before the clear text data. It was found that this data was consistent amongst several packet captures. After comparing the payload data found in the primary and secondary VSMs communications, it was found that both parties' communications during the configuration exchange had a similar format for the first 50 bytes of data. After further inspection of the data, it was found that the first 8 bytes of each packet's payload contained "03802b02". This was then followed by the recipient's MAC address and then the source's MAC address. The remaining data consisted of fixed and variable fields. Some of the fields incremented by one, while two of the fields decremented by one as each packet was sent. Other fields remained constant, keeping the same value in all of the packet captures. Table 4.1 reflects the findings after analyzing the packets sent during the clear text configuration communications.

Table 4.1.

Observations in the Clear Text Communications

Bytes	Observation
0-3	Fixed value, 03802b02
4-9	Recipient's MAC address
10-15	Sender's MAC address
16-19	Fixed value, 08004500
20-23	Incrementing value
24-27	Fixed value, 00004011
28-29	Decrementing value
30-43	Fixed value, unique to each VSM
44-45	Decrementing value
46-49	Incrementing value

The final test of the distributed communications involved resending captured communications. This was accomplished through the use of tcpreplay. For this test, the primary VSMs communication of the configuration was replayed to the secondary VSM. By capturing packets, it was found that the secondary VSM was responding in a similar fashion as it had during the original communications. The only difference found was that the responses contained different values in the incrementing and decrementing fields. Figure 4.31 depicts the replayed communications and the secondary VSM's response to the replayed communications.

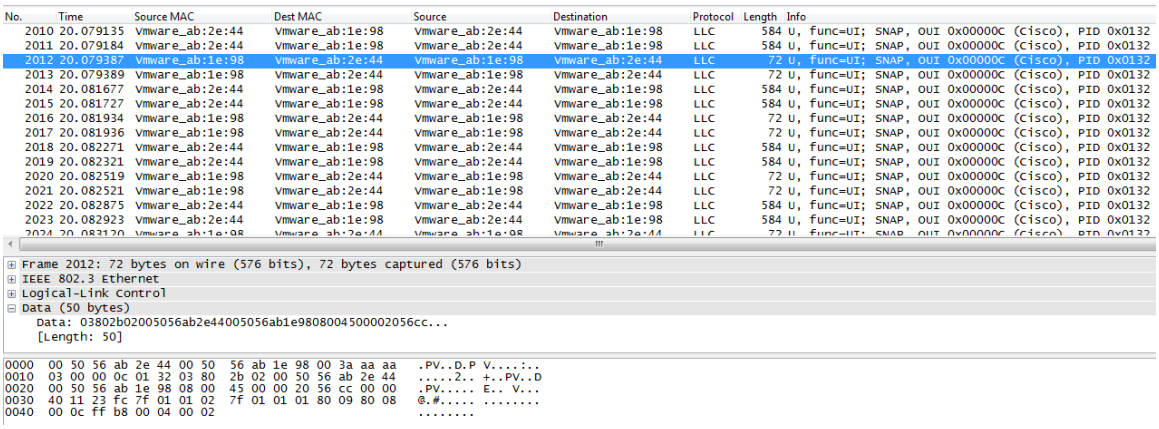

Figure 4.31. Packet capture depicting the replaying of the captured clear text configuration packets.

Although the secondary VSM appeared to be acknowledging the replayed communications, it was unclear whether it was accepting the replayed communications. It was found that due to command restrictions imposed on secondary VSMs, it was impossible to issue the command “show running-configuration”, which would normally be used to verify configurations. Due to this limitation, it was necessary to verify the configuration through other means. This was accomplished by repeatedly replaying configuration communications that differed from the configuration currently being used. As these communications were continuously replayed, the primary VSM was powered off, causing the secondary VSM to become the primary VSM. After the switchover, it was found that the legitimate configuration was still in use by the switched over VSM.

4.2.4. Distributed Switch Communications Conclusions

This research's investigation significantly improved the understanding of the communications used to facilitate the Nexus 1000V's distributed switching functionality. This is especially true since there was very little information available about the inner-workings of this switch. After assessing the results, the communication patterns used amongst the VSMs and the VEMs were learned. The results of this research also pointed to a potential security implication in the Nexus 1000V.

It was found that the VSMs and the VEMs communicate with one another over the Control VLAN. A proprietary protocol that operates at the data link layer is used by the switches to carry out these communications. Unfortunately, since the protocol is proprietary, information about it extremely limited. Nonetheless, the communications were still analyzed and it was found that the primary VSM maintains regular communications with the VEMs through the use of broadcasted packets. It was found that upon receiving the broadcast packets, the VEMs would respond to the primary VSM apparently acknowledging they had received the packet. This exchange would terminate with the primary VSM responding to each of the VEMs that had responded to it. It was also noticed that the VEMs did not communicate with one another. When configuration changes were made, it appeared as though the primary VSM would only communicate these changes to VEMs that were affected by the change. In other words, the VEMs only contained port-profile configuration information about the port-profiles in use on the particular VEM. It was also found that the primary VSM to VEM communications were likely encrypted. The basis for this was that the packets sent by the VSM to the VEM had seemingly random data with no identifiable patterns.

Like the primary VSM to VEM communications, it was found that the primary and secondary VSMs would communicate with one another using a data link protocol. It was also found that the VSMs were in constant communication with one another. This was likely a heartbeat check, that would allow the secondary VSM to recognize if the primary VSM had become unavailable, allowing it to promptly take over the primary VSMs role. When configuration changes were made, it was found that the primary VSM would transmit its entire running configuration file to the secondary VSM in clear text. This practice could mean that an attacker that had access to the Control VLAN could learn the VSM's configuration, making it easier for them to find other potential security issues. After analyzing several packet captures with this configuration exchange, several observations were made in regards to the potential fields used by the proprietary protocol during the transmissions of the configuration. It was also found that when packets previously sent by the primary VSM were replayed to the secondary VSM, the secondary VSM would respond with packets similar to those that were used when the configuration

was initially sent. The only noticeable difference was in the fields that had been found to found to increment or decrement in the other packet captures.

After exploring the communication mechanisms employed by the Nexus 1000V to facilitate its distributed switching, no security implications were found in the communications between the primary VSM and the VEMs and one potential security implication was found in the communications used between the VSMs. Like the security issues found in the physical switch vulnerabilities section, the security implications related to the distributed communications could be mitigated by configuring the network correctly. To prevent an unauthorized person from being able to eavesdrop on configuration communications being sent on the control VLAN, it should be ensured that the Control VLAN is used for nothing other than the communications between the VSMs and VEMs.

4.3. Virtual Machine Manipulation

The final test carried out, involved looking at the ramifications of having the VSMs as virtual machines. In particular, the ability to add and duplicate VSMs was assessed. This was done by first looking the security mechanisms used to allow a VSM to connect to an existing VSM. Second, it took a look at what happened if a standalone VSM were to be duplicated through vCenter. The third step involved looking at the effects of duplicating a primary VSM. The fourth and final step looked into the effects of duplicating a secondary VSM.

When assessing the processes of adding a VSM to vCenter, it was found that there were security mechanisms in place meant to prevent an unauthorized person from doing so. In order to add a new VSM to vCenter, a person is required to know the IP address of vCenter, as well as have administrative access to vCenter. In other words, a person would need to not only know the IP address of the server but also would need to have login credentials with administrative access. Once inside vCenter, it would then be possible to duplicate or even delete the VSM. It was found that additional information was necessary to create a new secondary VSM that would connect to the primary VSM. In addition to needing the IP address of vCenter and login credentials, one attempting to add a

secondary VSM must also know the primary VSM's IP address and the primary VSM's administrative login credentials.

4.3.1. Standalone VSM Duplication

After examining the security mechanisms preventing unauthorized VSM creation, the effects of duplicating a VSM were assessed. Initially a standalone VSM was duplicated through vCenter. It was found that network functionality on the virtual machines continued unhindered. Even though the network functionality provided to the virtual machines remained unmolested, the original VSM was aware of the duplicate VSM and logged a message every minute and forty seconds. It was ultimately found that the VSM was detecting the duplicate VSM and that there was another machine using the address as it. Figure 4.32 depicts a subset of the messages that were logged while the duplicate standalone VSM was present.

```
2012 Apr  1 15:23:41 Nexus100V %KERN-1-SYSTEM_MSG: Dropping received frames from duplicate VSM saddr (0x1020000) - kernel
2012 Apr  1 15:23:51 Nexus100V %ARP-2-DUP_SRCIP:  arp [2683]  Source address of packet received from 0050.56ab.1e99 on mgmt0 is duplicate of local, 10.19.35.99
2012 Apr  1 15:24:51 Nexus100V %ARP-2-DUP_SRCIP:  arp [2683]  Source address of packet received from 0050.56ab.1e99 on mgmt0 is duplicate of local, 10.19.35.99
2012 Apr  1 15:25:21 Nexus100V %ARP-2-DUP_SRCIP:  arp [2683]  Source address of packet received from 0050.56ab.1e99 on mgmt0 is duplicate of local, 10.19.35.99
2012 Apr  1 15:25:21 Nexus100V %KERN-1-SYSTEM_MSG: Dropping received frames from duplicate VSM saddr (0x1020000) - kernel
2012 Apr  1 15:25:51 Nexus100V %ARP-2-DUP_SRCIP:  arp [2683]  Source address of packet received from 0050.56ab.1e99 on mgmt0 is duplicate of local, 10.19.35.99
2012 Apr  1 15:26:27 Nexus100V %ARP-2-DUP_SRCIP:  arp [2683]  Source address of packet received from 0050.56ab.1e99 on mgmt0 is duplicate of local, 10.19.35.99
2012 Apr  1 15:27:02 Nexus100V %ARP-2-DUP_SRCIP:  arp [2683]  Source address of packet received from 0050.56ab.1e99 on mgmt0 is duplicate of local, 10.19.35.99
2012 Apr  1 15:27:02 Nexus100V %KERN-1-SYSTEM_MSG: Dropping received frames from duplicate VSM saddr (0x1020000) - kernel
2012 Apr  1 15:27:11 Nexus100V %ARP-2-DUP_SRCIP:  arp [2683]  Source address of packet received from 0050.56ab.1e99 on mgmt0 is duplicate of local, 10.19.35.99
```

Figure 4.32. Messages logged during the presence of a duplicate standalone VSM.

4.3.2. Primary VSM Duplication

Following the duplication of the standalone VSM, the test architecture was reverted to high availability mode and the secondary VSM was powered back on. Once high availability mode had been restored, the primary VSM was duplicated in the same manner as the previous test. Once again, it was found that the virtual machine's network connectivity was unhindered. After looking at the switches logs, the warnings depicted in Figure 4.33 were found.

```
2012 Mar 31 12:19:18 Nexus100V %KERN-1-SYSTEM_MSG: Dropping received frames from duplicate VSM saddr (0x1020000) - kernel
2012 Mar 31 12:19:23 Nexus100V %ARP-2-DUP_SRCIP:  arp [2683]  Source address of packet received from 0050.56ab.1e99 on mgmt0 is duplicate of local, 10.19.35.99
2012 Mar 31 12:20:09 Nexus100V %ARP-2-DUP_SRCIP:  arp [2683]  Source address of packet received from 0050.56ab.1e99 on mgmt0 is duplicate of local, 10.19.35.99
2012 Mar 31 12:20:58 Nexus100V %ARP-2-DUP_SRCIP:  arp [2683]  Source address of packet received from 0050.56ab.1e99 on mgmt0 is duplicate of local, 10.19.35.99
2012 Mar 31 12:20:58 Nexus100V %KERN-1-SYSTEM_MSG: Dropping received frames from duplicate VSM saddr (0x1020000) - kernel
2012 Mar 31 12:21:09 Nexus100V %ARP-2-DUP_SRCIP:  arp [2683]  Source address of packet received from 0050.56ab.1e99 on mgmt0 is duplicate of local, 10.19.35.99
2012 Mar 31 12:22:09 Nexus100V %ARP-2-DUP_SRCIP:  arp [2683]  Source address of packet received from 0050.56ab.1e99 on mgmt0 is duplicate of local, 10.19.35.99
2012 Mar 31 12:22:38 Nexus100V %ARP-2-DUP_SRCIP:  arp [2683]  Source address of packet received from 0050.56ab.1e99 on mgmt0 is duplicate of local, 10.19.35.99
2012 Mar 31 12:22:38 Nexus100V %KERN-1-SYSTEM_MSG: Dropping received frames from duplicate VSM saddr (0x1020000) - kernel
2012 Mar 31 12:23:09 Nexus100V %ARP-2-DUP_SRCIP:  arp [2683]  Source address of packet received from 0050.56ab.1e99 on mgmt0 is duplicate of local, 10.19.35.99
2012 Mar 31 12:24:09 Nexus100V %ARP-2-DUP_SRCIP:  arp [2683]  Source address of packet received from 0050.56ab.1e99 on mgmt0 is duplicate of local, 10.19.35.99
2012 Mar 31 12:24:18 Nexus100V %KERN-1-SYSTEM_MSG: Dropping received frames from duplicate VSM saddr (0x1020000) - kernel
```

Figure 4.33. Messages logged during the presence of a duplicate primary VSM.

4.3.3. Secondary VSM Duplication

The final test looking at the manipulation of the VSM virtual machines sought to duplicate a secondary VSM. This test was carried out after the duplicate primary VSM had been deleted and the architecture was back to its original state. Like in the other two tests discussed in this section, the duplication of the VSM did not affect the network connectivity of the virtual machines. Instead, the only noticeable result was found within the logs of the VSM, where it was noted that another machine was sharing the same address. Figure 4.34 depicts a selection of the errors that were found in the VSM's log.

```
2012 Apr  1 12:39:36 Nexus100V %ARP-2-DUP_SRCIP:  arp [3057]  Source address of packet received from 0050.56ab.1ea9 on mgmt0 is duplicate of local, 10.19.35.99
2012 Apr  1 12:39:38 Nexus100V %ARP-2-DUP_SRCIP:  arp [3057]  Source address of packet received from 0050.56ab.1ea9 on mgmt0 is duplicate of local, 10.19.35.99
2012 Apr  1 12:44:38 Nexus100V %ARP-2-DUP_SRCIP:  arp [3057]  Source address of packet received from 0050.56ab.1ea9 on mgmt0 is duplicate of local, 10.19.35.99
2012 Apr  1 12:47:38 Nexus100V %ARP-2-DUP_SRCIP:  arp [3057]  Source address of packet received from 0050.56ab.1ea9 on mgmt0 is duplicate of local, 10.19.35.99
2012 Apr  1 13:03:39 Nexus100V %ARP-2-DUP_SRCIP:  arp [3057]  Source address of packet received from 0050.56ab.1ea9 on mgmt0 is duplicate of local, 10.19.35.99
2012 Apr  1 13:10:46 Nexus100V %ARP-2-DUP_SRCIP:  arp [3057]  Source address of packet received from 0050.56ab.1ea9 on mgmt0 is duplicate of local, 10.19.35.99
2012 Apr  1 13:11:40 Nexus100V %ARP-2-DUP_SRCIP:  arp [3057]  Source address of packet received from 0050.56ab.1ea9 on mgmt0 is duplicate of local, 10.19.35.99
2012 Apr  1 13:16:16 Nexus100V %ARP-2-DUP_SRCIP:  arp [3057]  Source address of packet received from 0050.56ab.1ea9 on mgmt0 is duplicate of local, 10.19.35.99
```

Figure 4.34. Messages logged during the presence of a duplicate secondary VSM

4.3.4. Virtual Machine Manipulation Conclusions

Upon completion of this section of the research, a better understanding of effects of duplicating a VSM had been gained. It was found that in all three of the tested scenarios the network functionality provided to the virtual machines remained unaffected, despite the creation of the duplicate VSMs. The only noticeable implications were found within the log of the VSM. Furthermore, it was found that in order for this attack to take place, it would be necessary for an attacker to have administrative access to vCenter. Because of these findings, it was determined that the duplication of the VSM virtual machines does not present itself as a security implication.

4.4. Summary and Conclusions

In this section, the results of this research's experiments were presented and conclusions that could be drawn from these results were discussed. The differences in the Nexus 1000V's vulnerability in comparison to physical switches was better understood. Knowledge and potential issues with the distributed communications between the primary and secondary VSMs was also gained. It also pointed out the effects of creating duplicate instances of the VSMs. Finally, the implications these results have were discussed in detail.

With respect to results of the tests carried out in this research, it can be concluded that if proper configuration considerations are made, the Nexus 1000V does not present additional security implications in comparison to physical switches. In fact, it can be said that it presents fewer issues as its independence of STP means there is no concern of STP manipulation. That being said, if proper configurations are not made, the Nexus 1000V will be susceptible to CAM overflows, VLAN hopping, ARP poisoning, and Private VLAN vulnerabilities. In addition to these vulnerabilities that are commonly found in physical switches, if the Nexus 1000V is not configured correctly, an attacker could gain access to the Nexus 1000V's configuration that is being sent from the primary VSM to the secondary VSM in clear text. With respect to the creation of additional VSMs, there was no observed reason to be concerned about rogue VSMs being added to the network because of the need for administrative access to vCenter. The tests in this research also showed that even if VSMs are duplicated, the effects are only noticeable within the logs of the Nexus 1000V and the network connectivity provided to the virtual machines is unaffected.

These conclusions mean, if proper configuration considerations are made, organizations wishing use virtualization, whether it for lessening the environmental impact of their datacenters or for any other reason, no longer need to sacrifice network functionality. They can now use the Nexus 1000V to provide the virtual machines with robust network functionality without creating additional security concerns. This functionality provided by the Nexus 1000V gives the virtual machines with the same network functionality that is offered by physical switches but has been absent in

traditional virtual switches. It also provides its users with a configuration interface that network administrators are familiar with, as it is the same that has been used to configure physical switches. This familiar interface allows network administrators to use the same security practices and configurations that they have previously used with their physical switches. In conclusion, if configuration recommendations are heeded, the Cisco Nexus 1000V does not present any additional security implications with respect to physical switches; however, if the configuration recommendations go unheeded, the security of the Nexus 1000V could be affected by the switch configuration being sent in clear text, in addition to other security implications that affect physical switches.

CHAPTER 5. FUTURE WORK

With the completion of this research, many results had been gathered, and conclusions were drawn from these results. Despite this, there were still multiple areas for future work. These areas stem from the information learned during the research, the scope of this research and the limitations of the resources available for this research.

The first area for future work is to further explore the clear-text communications being sent from the primary VSM to the secondary VSM. Although these clear-text communications were captured and patterns in the protocol used were noted, the exact purpose of the observed fields was not known. There is still much information about this protocol to be determined. Such information could potentially be used to manipulate the Nexus 1000V. There is also potential that the replay attempts from this research could be improved if a better understanding of the protocol is gained.

The second area for future work involves analyzing the communications between the VSM to the VEMs. These communications appeared to be encrypted; however, the standard of encryption being employed remained unknown. Should this be discovered, there is potential that there is an inherent vulnerability in the encryption being used. Furthermore, there is potential that the encryption is not being implemented, negating the security offered by the encryption.

The third area of for future work involves assessing the security of the Nexus 1000V in a datacenter environment. Although the results of this research carried out should be representative of those found in a datacenter, there is potential that additional issues will be introduced or made more apparent as the deployment size of the Nexus 1000V is scaled. There is also potential that additional vCenter components could influence the functionality of the Nexus 1000V.

The final area for future work involves using a different physical switch. In particular, a non-Cisco physical switch should be used. Although theoretically other vendor's equipment should work with the Nexus 1000V, there is potential that the slight differences in functionality could create additional security implications.

REFERENCES

REFERENCES

B4rtm4n. (2005). *Hex2bin*. Retrieved from http://r00tsecurity.org/forums/topic/10698-crafting-routing-protocols-using-nemesis/

Bastien, G., Nasseh, S., & Degu, C. (2006). *CCSP self-study: CCSP SNRS exam certification guide* (pp. 279-302). Indianapolis, IN: Cisco Press.

Bruschi, D. Ornaghi, A., & Rosti, E. (2003). S-ARP: A secure address resolution protocol. *Proceedings of the Computer Security Applications Conference, Las Vegas, NV, 19,* 66-74. doi:10.1109/CSAC.2003.1254311

Cisco. (2009). *Cisco VN-Link: virtualization-aware networking*. Retrieved from http://www.cisco.com/en/US/solutions/collateral/ns340/ns517/ns224/ns892/ns894/white_paper_c11-525307.pdf

Cisco. (2010). *Virtual machine networking: standards and solutions*. Retrieved from http://www.cisco.com/en/US/prod/collateral/switches/ps9441/ps9902/whitepaper_c11-620065.pdf

Cisco. (2011a) *Cisco Nexus 1000V getting started guide, release 4.2(1) sv1(4).* Retrieved From http://www.cisco.com/en/us/docs/switches/datacenter/nexus1000/sw/4_2_1_s_v_1_4/getting_started/configuration/guide/n1000v_gsg.pdf

Cisco. (2011b). *Cisco Nexus 1000V release notes, release 4.0(4) SV1(3c)*. Retrieved from http://www.cisco.com/en/US/docs/switches/datacenter/nexus1000/sw/4_0_4_s_v_1_3_c/release/notes/n1000v_rn.pdf

Cisco. (2012). *Cisco Nexus 1000V installation and configuration guide, release 4.2(1)SV1(5.1)*. Retrieved from http://www.cisco.com/en/US/docs/switches/datacenter/nexus1000/sw/4_2_1_s_v_1_5_1/install_upgrade/vsm_vem/guide/n1000v_installupgrade.pdf

Davoli, R. VDE: virtual distributed ethernet. (2005) *Proceedings of the First International Conference on Testbeds and Research Infrastructures for the Devlopment of Network and Communities*. The Institute for Computer Sciences, Social Informatics and Telecommunications Engineering,Trento, Italy.

Farrow, R. (2003). VLANs: virtually insecure?. *Network Magazine, 18*(3), 62-63.

Froom, R., Sivasubramanian, B., Frahim, E., & Houston, T. (2007). *Authorized self-study guide: Building cisco multilayer switched networks (BCMSN)* (4th ed.). Indianapolis: Cisco Press.

Goldman, J. E., & Rawles, P. T. (2004). *Applied data communications: A business-oriented approach* (pp. 58-61). Hoboken, NJ: John Wiley and Sons.

Luo, Y., Murray, E., & Ficarra, T. (2010). Accelerated virtual switching with programmable NICs for scalable data center networking. *Proceedings from the Virtualized Infrastructure Systems and Architectures 2010 Workshop*. Association for Computer Machinery, New Delhi, India.

Mipetrin. (2010). *Re: what port/protocol does the MTS service use between active and standby VSMs?*. Retrieved from http://communities.cisco.com/message/59161#59161

Nanda, N., & Chiueh, T. (2005, February). *A survey on virtualization technologies*. Retrieved from Experimental Computer Systems Lab: http://www.ecsl.cs.sunysb.edu/tr/TR179.pdf

Nathan, J. (n.d.). *Nemesis.* Retrieved from http://nemesis.sourceforge.net/

Omella, A. & Berrueta D. (n.d.). *Yersinia*. Retrieved from http://www.yersinia.net/

Ornaghi, A. & Valleri, M. (2005, May 5). Ettercap. Retrieved from http://ettercap.sourceforge.net/index.php

Pettit, J., Gross, J., Pfaff, B., Casado, M., & Crosby, S. (2010). Virtual switching in an era of advanced edges. *Proceedings from the 2nd Workshop on Data Center – Converged and Virtual Ethernet Switching*. The International Telegraffic Congress, Amsterdam, Netherlands.

Smith, J., & Nair, R. (2005). The architecture of virtual machines. *Computer, 38*(5), 32-38.

VMware. (2009). *Introduction to VMware vSphere*. Retrieved from http://www.vmware.com/pdf/vsphere4/r40/vsp_40_intro_vs.pdf

Song, D. (n.d.). *Macof(8) - Linux man page*. Retrieved from http://linux.die.net/man/8/macof

Turner, A. (2010). *Tcpreplay(1) – Linux man page*. Retreived from http://linux.die.net/man/1/tcpreplay

Vaporub. (2009). *Generat a random MAC address*. Retrieved from http://www.commandlinefu.com/commands/view/745/generat-a-random-mac-address

Vyncke E. & Paggen C. (2008). *LAN switch security: what hackers know about your switches* (pp. 54-64). Indianapolis, IN: Cisco Press.

Zhou, S. (2010). Virtual networking. *ACM SIGOPS Operating System Review, 44*(4), 80-85.

APPENDICES

Appendix A Physical Switch Configuration

```
version 12.2
no service pad
service timestamps debug uptime
service timestamps log uptime
service password-encryption
!
hostname Switch
!
enable secret 5 $1$zeOn$DBN.J.seCiJkN2xC8B6ij1
!
username admin password 7 08205C4158480A4641
aaa new-model
aaa authentication login loc local
!
aaa session-id common
switch 1 provision ws-c3750-24ts
vtp mode transparent
ip subnet-zero
ip domain-name 555.cit.lcl
!
no file verify auto
```

```
!
spanning-tree mode mst
spanning-tree extend system-id
no spanning-tree vlan 300
!
vlan internal allocation policy ascending
vlan dot1q tag native
!
vlan 971
 name Control
!
vlan 972
 name Packet
!
vlan 973
 name secureVlan
!
vlan 974
 name privateVLAN
!
vlan 1935
 name Management
!
```

```
interface FastEthernet1/0/1
 description ESXI Host 10 - MGMT
 switchport access vlan 1935
 switchport mode access
 switchport nonegotiate
 no cdp enable
!
interface FastEthernet1/0/2
 switchport mode access
 switchport nonegotiate
 no cdp enable
!
interface FastEthernet1/0/3
 description ESXI Host 11 - MGMT
 switchport access vlan 1935
 switchport trunk encapsulation dot1q
 switchport mode access
 switchport nonegotiate
 no cdp enable
!
interface FastEthernet1/0/4
 switchport mode access
 switchport nonegotiate
```

```
 no cdp enable
!
interface FastEthernet1/0/5
 description ESXI Host 12 - MGMT
 switchport access vlan 1935
 switchport mode access
 switchport nonegotiate
 no cdp enable
!
interface FastEthernet1/0/6
 switchport mode access
 switchport nonegotiate
 no cdp enable
!
interface FastEthernet1/0/7
 description Network Sniffer
 switchport mode access
 switchport nonegotiate
 no cdp enable
!
interface FastEthernet1/0/8
 switchport mode access
 switchport nonegotiate
```

```
 no cdp enable
!
interface FastEthernet1/0/9
 description ESXI/vCENTER
 switchport access vlan 1935
 switchport mode access
 switchport nonegotiate
 no cdp enable
!
interface FastEthernet1/0/10
 switchport mode access
 switchport nonegotiate
 no cdp enable
!
interface FastEthernet1/0/11
 switchport trunk encapsulation dot1q
 switchport mode trunk
 switchport nonegotiate
 no cdp enable
!
interface FastEthernet1/0/12
 switchport mode access
 switchport nonegotiate
```

```
 no cdp enable
!
interface FastEthernet1/0/13
 description ESXI Host 10 - dvSwitch
 switchport trunk encapsulation dot1q
 switchport mode trunk
 switchport nonegotiate
!
interface FastEthernet1/0/14
 description ESXI HOST 14 - MGMT
 switchport access vlan 1935
 switchport mode access
 switchport nonegotiate
 no cdp enable
!
interface FastEthernet1/0/15
 description ESXI Host 11 - dvSwitch
 switchport trunk encapsulation dot1q
 switchport mode trunk
 switchport nonegotiate
!
interface FastEthernet1/0/16
 switchport mode access
```

```
 switchport nonegotiate
 no cdp enable
!
interface FastEthernet1/0/17
 description ESXI Host 12 - dvSwitch
 switchport trunk encapsulation dot1q
 switchport mode trunk
 switchport nonegotiate
!
interface FastEthernet1/0/18
 switchport mode access
 switchport nonegotiate
 no cdp enable
!
interface FastEthernet1/0/19
 description ESXI Host 14 - dvSwitch
switchport trunk encapsulation dot1q
 switchport mode trunk
 switchport nonegotiate
 no cdp enable
!
interface FastEthernet1/0/20
 switchport mode access
```

```
 switchport nonegotiate
 no cdp enable
!
interface FastEthernet1/0/21
 description SnifferMgmt
 switchport access vlan 1935
 switchport mode access
 switchport nonegotiate
 no cdp enable
!
interface FastEthernet1/0/22
switchport mode access
 switchport nonegotiate
 no cdp enable
!
interface FastEthernet1/0/23
 switchport mode access
 switchport nonegotiate
 no cdp enable
!
interface FastEthernet1/0/24
 description UPLINK
 switchport trunk encapsulation dot1q
```

```
 switchport trunk allowed vlan 971-974,1935
 switchport mode trunk
 switchport nonegotiate
 no cdp enable
!
interface GigabitEthernet1/0/1
!
interface GigabitEthernet1/0/2
!
interface Vlan1

 no ip route-cache
!
no ip route-cache
!
interface Vlan971
 ip address 10.97.1.2 255.255.255.0
!
interface Vlan972
 ip address 10.97.2.2 255.255.255.0
!
interface Vlan973
 no ip address
```

```
!
interface Vlan974
 description protectedvlan
 ip address 10.97.4.3 255.255.255.0
!
interface Vlan1935
 description management_access
 ip address 10.19.35.2 255.255.255.0
!
ip default-gateway 10.19.35.1
ip classless
ip route 0.0.0.0 0.0.0.0 10.19.35.1
ip http server
ip http secure-server
!
radius-server source-ports 1645-1646
!
control-plane
!
line con 0
line vty 0 4
 login authentication loc
line vty 5 15
```

```
!
monitor session 1 source vlan 971
monitor session 1 destination interface Fa1/0/7
!
end
```

Appendix B VSM Configuration

```
version 4.2(1)SV1(5.1)
no feature telnet
feature private-vlan

username admin password 5 $1$EyzSqiwo$EE7qs0xQTT80dpVWOlhaZ.  role network-
admin
username ben password 5 $1$P3nTU.oq$EttRTOP/zFqOxSvDskr3y.  role network-
operator

banner motd #Nexus 1000v Switch#

ip domain-lookup
hostname Nexus100V
vlan dot1Q tag native
system default switchport
logging event link-status default
vem 3
  host vmware id 44454c4c-5300-1043-8036-b9c04f304731
vem 4
  host vmware id 44454c4c-4e00-1037-8044-b7c04f574331
vem 5
```

```
  host vmware id 44454c4c-5600-1038-8044-c7c04f574331
snmp-server user ben network-operator auth md5
0x1800e04e7dcfa9c4906ed37a9659fb30 priv 0x1800e04e7dcfa9c4906ed37a9659fb30
localizedkey
snmp-server user admin network-admin auth md5
0x1800e04e7dcfa9c4906ed37a9659fb30 priv 0x1800e04e7dcfa9c4906ed37a9659fb30
localizedkey

vrf context management
  ip route 0.0.0.0/0 10.19.35.1
vlan 1,971-974,1935
vlan 1
vlan 971
  name Control
vlan 972
  name Packet
vlan 973
  name SecureVLAN
vlan 974
  name PrivateVLAN
vlan 1935
  name Management
```

```
port-channel load-balance ethernet source-mac
port-profile default max-ports 32
port-profile type ethernet Unused_Or_Quarantine_Uplink
  vmware port-group
  shutdown
  description Port-group created for Nexus1000V internal usage. Do not use.
  state enabled
port-profile type vethernet Unused_Or_Quarantine_Veth
  vmware port-group
  shutdown
  description Port-group created for Nexus1000V internal usage. Do not use.
  state enabled
port-profile type vethernet n1kv-system-control
  vmware port-group
  switchport mode access
  switchport access vlan 971
  no shutdown
  system vlan 971
  state enabled
port-profile type vethernet n1kv-system-management
  vmware port-group
  switchport mode access
  switchport access vlan 1935
```

```
  no shutdown
  system vlan 1935
  state enabled
port-profile type vethernet n1kv-system-packet
  vmware port-group
  switchport mode access
  switchport access vlan 972
  no shutdown
  system vlan 972
  state enabled
port-profile type ethernet n1kv-uplink0
  vmware port-group
  switchport mode trunk
  switchport trunk allowed vlan 1,971-974,1935
  switchport trunk native vlan 1
  channel-group auto mode on mac-pinning
  no shutdown
  system vlan 971-974,1935
  state enabled
port-profile type vethernet secureVlan
  vmware port-group
  switchport access vlan 973
  no shutdown
```

```
  description second data vlan
  state enabled
port-profile type vethernet ProtectedVLAN
  vmware port-group
  switchport access vlan 1
  switchport mode access
  no shutdown
  description Protected data vlan
  state enabled
port-profile type vethernet VirtualSniffer
  vmware port-group
  switchport mode access
  no shutdown
  description Virtual Sniffer
  state enabled
port-profile type vethernet Native
  vmware port-group
  switchport access vlan 1
  no shutdown
  state enabled


system storage-loss log time 30
vdc Nexus100V id 1
```

```
  limit-resource vlan minimum 16 maximum 2049
  limit-resource monitor-session minimum 0 maximum 2
  limit-resource vrf minimum 16 maximum 8192
  limit-resource port-channel minimum 0 maximum 768
  limit-resource u4route-mem minimum 1 maximum 1
  limit-resource u6route-mem minimum 1 maximum 1
  limit-resource m4route-mem minimum 58 maximum 58
  limit-resource m6route-mem minimum 8 maximum 8



interface port-channel1
  inherit port-profile n1kv-uplink0
  vem 3

interface port-channel2
  inherit port-profile n1kv-uplink0
  vem 4

interface port-channel3
  inherit port-profile n1kv-uplink0
  vem 5

interface mgmt0
```

```
  ip address 10.19.35.99/24

interface Vethernet1
  inherit port-profile n1kv-system-control
  description Nexus1000v, Network Adapter 1
  vmware dvport 64 dvswitch uuid "73 40 2b 50 c6 ff f6 e0-fc 81 34 cb 42 63 19 6c"
  vmware vm mac 0050.56AB.2E44

interface Vethernet2
  inherit port-profile n1kv-system-management
  description Nexus1000v, Network Adapter 2
  vmware dvport 100 dvswitch uuid "73 40 2b 50 c6 ff f6 e0-fc 81 34 cb 42 63 19 6c"
  vmware vm mac 0050.56AB.2E45

interface Vethernet3
  inherit port-profile n1kv-system-packet
  description Nexus1000v, Network Adapter 3
  vmware dvport 128 dvswitch uuid "73 40 2b 50 c6 ff f6 e0-fc 81 34 cb 42 63 19 6c"
  vmware vm mac 0050.56AB.2E46

interface Vethernet4
  inherit port-profile n1kv-system-management
  description VMware VMkernel, vmk0
```

```
  vmware dvport 101 dvswitch uuid "73 40 2b 50 c6 ff f6 e0-fc 81 34 cb 42 63 19 6c"
  vmware vm mac 0019.B934.8420

interface Vethernet5
  inherit port-profile n1kv-system-management
  description VMware VMkernel, vmk0
  vmware dvport 102 dvswitch uuid "73 40 2b 50 c6 ff f6 e0-fc 81 34 cb 42 63 19 6c"
  vmware vm mac 0019.B932.BF94

interface Vethernet6
  inherit port-profile n1kv-system-packet
  description CentOS Host 1 (101), Network Adapter 1
  vmware dvport 131 dvswitch uuid "73 40 2b 50 c6 ff f6 e0-fc 81 34 cb 42 63 19 6c"
  vmware vm mac 0050.56AB.2E2A

interface Vethernet7
  inherit port-profile n1kv-system-control
  description BackTrack (111), Network Adapter 1
  vmware dvport 66 dvswitch uuid "73 40 2b 50 c6 ff f6 e0-fc 81 34 cb 42 63 19 6c"
  vmware vm mac 0050.56AB.2E47

interface Vethernet8
  inherit port-profile n1kv-system-packet
```

```
  description CentOS Host 2 (102), Network Adapter 1
  vmware dvport 132 dvswitch uuid "73 40 2b 50 c6 ff f6 e0-fc 81 34 cb 42 63 19 6c"
  vmware vm mac 0050.56AB.563C

interface Vethernet9
  inherit port-profile n1kv-system-packet
  description BackTrack2, Network Adapter 1
  vmware dvport 133 dvswitch uuid "73 40 2b 50 c6 ff f6 e0-fc 81 34 cb 42 63 19 6c"
  vmware vm mac 0050.56AB.563B

interface Vethernet10
  inherit port-profile n1kv-system-packet
  description attack, Network Adapter 1
  vmware dvport 129 dvswitch uuid "73 40 2b 50 c6 ff f6 e0-fc 81 34 cb 42 63 19 6c"
  vmware vm mac 0050.56AB.2E40

interface Vethernet11
  inherit port-profile n1kv-system-packet
  description Virtual Sniffer 10.19.35.197, Network Adapter 1
  vmware dvport 135 dvswitch uuid "73 40 2b 50 c6 ff f6 e0-fc 81 34 cb 42 63 19 6c"
  vmware vm mac 0050.56AB.563E

interface Vethernet12
```

```
  inherit port-profile n1kv-system-management
  description Virtual Sniffe...19.35.197, Network Adapter 2
  vmware dvport 103 dvswitch uuid "73 40 2b 50 c6 ff f6 e0-fc 81 34 cb 42 63 19 6c"
  vmware vm mac 0050.56AB.563F

interface Vethernet13
  inherit port-profile n1kv-system-management
  description attack, Network Adapter 2
  vmware dvport 102 dvswitch uuid "73 40 2b 50 c6 ff f6 e0-fc 81 34 cb 42 63 19 6c"
  vmware vm mac 0050.56AB.1E7A

interface Vethernet14
  inherit port-profile VirtualSniffer
  description Virtual Sniffer 2, Network Adapter 1
  vmware dvport 640 dvswitch uuid "73 40 2b 50 c6 ff f6 e0-fc 81 34 cb 42 63 19 6c"
  vmware vm mac 0050.56AB.1E7B

interface Vethernet15
  inherit port-profile n1kv-system-management
  description Virtual Sniffer 2, Network Adapter 2
  vmware dvport 104 dvswitch uuid "73 40 2b 50 c6 ff f6 e0-fc 81 34 cb 42 63 19 6c"
  vmware vm mac 0050.56AB.1E7C
```

```
interface Vethernet16
  inherit port-profile n1kv-system-control
  description Virtual Sniffer 3, Network Adapter 1
  vmware dvport 65 dvswitch uuid "73 40 2b 50 c6 ff f6 e0-fc 81 34 cb 42 63 19 6c"
  vmware vm mac 0050.56AB.1E7D

interface Vethernet17
  inherit port-profile n1kv-system-management
  description Virtual Sniffer 3, Network Adapter 2
  vmware dvport 101 dvswitch uuid "73 40 2b 50 c6 ff f6 e0-fc 81 34 cb 42 63 19 6c"
  vmware vm mac 0050.56AB.1E7E

interface Vethernet18
  inherit port-profile secureVlan
  description CentOS Host 3, Network Adapter 1
  vmware dvport 480 dvswitch uuid "73 40 2b 50 c6 ff f6 e0-fc 81 34 cb 42 63 19 6c"
  vmware vm mac 0050.56AB.1E7F

interface Ethernet3/2
  inherit port-profile n1kv-uplink0

interface Ethernet4/1
  inherit port-profile n1kv-uplink0
```

```
interface Ethernet5/1
  inherit port-profile n1kv-uplink0


interface control0
line console
boot kickstart bootflash:/nexus-1000v-kickstart-mz.4.2.1.SV1.5.1.bin sup-1
boot system bootflash:/nexus-1000v-mz.4.2.1.SV1.5.1.bin sup-1
boot kickstart bootflash:/nexus-1000v-kickstart-mz.4.2.1.SV1.5.1.bin sup-2
boot system bootflash:/nexus-1000v-mz.4.2.1.SV1.5.1.bin sup-2
monitor session 1
  source vlan 972 rx
  source vlan 1 both
  destination interface Vethernet14
  no shut
monitor session 2
  no shut
svs-domain
  domain id 555
  control vlan 971
  packet vlan 972
  svs mode L2
svs connection vcenter
  protocol vmware-vim
```

```
  remote ip address 10.19.35.50 port 80
  vmware dvs uuid "73 40 2b 50 c6 ff f6 e0-fc 81 34 cb 42 63 19 6c" datacenter-name Lab
  max-ports 8192
  connect
vsn type vsg global
  tcp state-checks
vnm-policy-agent
  registration-ip 0.0.0.0
  shared-secret **********
```

Appendix C VMX Configuration Files

Backtrack Attack VMX Configuration File

```
.encoding = "UTF-8"
config.version = "8"
virtualHW.version = "8"
pciBridge0.present = "true"
pciBridge4.present = "true"
pciBridge4.virtualDev = "pcieRootPort"
pciBridge4.functions = "8"
pciBridge5.present = "true"
pciBridge5.virtualDev = "pcieRootPort"
pciBridge5.functions = "8"
pciBridge6.present = "true"
pciBridge6.virtualDev = "pcieRootPort"
pciBridge6.functions = "8"
pciBridge7.present = "true"
pciBridge7.virtualDev = "pcieRootPort"
pciBridge7.functions = "8"
vmci0.present = "true"
hpet0.present = "true"
nvram = "BT5R2-GNOME-VM-64.nvram"
virtualHW.productCompatibility = "hosted"
```

powerType.powerOff = "hard"

powerType.powerOn = "hard"

powerType.suspend = "hard"

powerType.reset = "hard"

displayName = "Attack 2"

extendedConfigFile = "BT5R2-GNOME-VM-64.vmxf"

vcpu.hotadd = "true"

scsi0.present = "true"

scsi0.sharedBus = "none"

scsi0.virtualDev = "lsilogic"

memsize = "768"

mem.hotadd = "true"

scsi0:0.present = "true"

scsi0:0.fileName = "BT5R2-GNOME-VM-64.vmdk"

scsi0:0.deviceType = "scsi-hardDisk"

sched.scsi0:0.shares = "normal"

sched.scsi0:0.throughputCap = "off"

ide1:0.present = "true"

ide1:0.fileName = "No Devices available"

ide1:0.deviceType = "atapi-cdrom"

ide1:0.startConnected = "false"

usb.present = "true"

ehci.present = "true"

guestOS = "ubuntu"

uuid.bios = "56 4d b2 59 29 b8 d0 67-de ef 09 22 8c 26 77 5e"

vc.uuid = "50 2b 1e 13 58 b6 3a f3-a5 f4 f6 2d 4b a2 9f fd"

snapshot.action = "keep"

sched.cpu.min = "0"

sched.cpu.units = "mhz"

sched.cpu.shares = "normal"

sched.mem.min = "0"

sched.mem.shares = "normal"

tools.upgrade.policy = "manual"

usb.vbluetooth.startConnected = "TRUE"

replay.supported = "FALSE"

unity.wasCapable = "FALSE"

replay.filename = ""

scsi0:0.redo = ""

pciBridge0.pciSlotNumber = "17"

pciBridge4.pciSlotNumber = "21"

pciBridge5.pciSlotNumber = "22"

pciBridge6.pciSlotNumber = "23"

pciBridge7.pciSlotNumber = "24"

scsi0.pciSlotNumber = "16"

usb.pciSlotNumber = "32"

ehci.pciSlotNumber = "35"

vmci0.pciSlotNumber = "36"

usb:1.present = "TRUE"

tools.remindInstall = "FALSE"

vmotion.checkpointFBSize = "4194304"

usb:1.speed = "2"

usb:1.deviceType = "hub"

usb:1.port = "1"

usb:1.parent = "-1"

ethernet0.present = "TRUE"

ethernet0.networkName = ""

ethernet0.addressType = "vpx"

ethernet0.generatedAddress = "00:50:56:ab:2e:47"

ethernet0.dvs.switchId = "73 40 2b 50 c6 ff f6 e0-fc 81 34 cb 42 63 19 6c"

ethernet0.dvs.portId = "66"

ethernet0.dvs.portgroupId = "dvportgroup-185"

ethernet0.dvs.connectionId = "1806113617"

vmci0.id = "-1943636130"

tools.syncTime = "FALSE"

uuid.location = "56 4d e1 40 c6 ba 29 49-f0 8a 27 dd 49 9a ac f1"

cleanShutdown = "FALSE"

sched.swap.derivedName = "/vmfs/volumes/4f58d85a-cf6241e0-77de-

0019b9348420/BackTrack/BT5R2-GNOME-VM-64-5954086a.vswp"

ethernet0.pciSlotNumber = "33"

hostCPUID.0 = "0000000a756e65476c65746e49656e69"

hostCPUID.1 = "000006f2000208000000e3bdbfebfbff"

hostCPUID.80000001 = "00000000000000000000000120000800"

guestCPUID.0 = "0000000a756e65476c65746e49656e69"

guestCPUID.1 = "000006f200010800800022010febfbff"

guestCPUID.80000001 = "00000000000000000000000120000800"

userCPUID.0 = "0000000a756e65476c65746e49656e69"

userCPUID.1 = "000006f2000208000000e3bdbfebfbff"

userCPUID.80000001 = "00000000000000000000000120000800"

evcCompatibilityMode = "FALSE"

floppy0.present = "FALSE"

usb:0.present = "TRUE"

usb:0.deviceType = "hid"

usb:0.port = "0"

usb:0.parent = "-1"

CentOS Attack VMX Configuration File

.encoding = "UTF-8"

config.version = "8"

virtualHW.version = "7"

pciBridge0.present = "true"

pciBridge4.present = "true"

pciBridge4.virtualDev = "pcieRootPort"

pciBridge4.functions = "8"

pciBridge5.present = "true"

pciBridge5.virtualDev = "pcieRootPort"

pciBridge5.functions = "8"

pciBridge6.present = "true"

pciBridge6.virtualDev = "pcieRootPort"

pciBridge6.functions = "8"

pciBridge7.present = "true"

pciBridge7.virtualDev = "pcieRootPort"

pciBridge7.functions = "8"

vmci0.present = "true"

nvram = "CentOS 2.nvram"

virtualHW.productCompatibility = "hosted"

powerType.powerOff = "soft"

powerType.powerOn = "hard"

powerType.suspend = "hard"

```
powerType.reset = "soft"
displayName = "Attack"
extendedConfigFile = "CentOS 2.vmxf"
floppy0.present = "true"
scsi0.present = "true"
scsi0.sharedBus = "none"
scsi0.virtualDev = "lsilogic"
memsize = "256"
scsi0:0.present = "true"
scsi0:0.fileName = "CentOS 2.vmdk"
scsi0:0.deviceType = "scsi-hardDisk"
sched.scsi0:0.shares = "normal"
sched.scsi0:0.throughputCap = "off"
ide1:0.present = "true"
ide1:0.deviceType = "cdrom-image"
floppy0.startConnected = "false"
floppy0.fileName = ""
floppy0.clientDevice = "true"
ethernet0.present = "true"
ethernet0.dvs.switchId = "73 40 2b 50 c6 ff f6 e0-fc 81 34 cb 42 63 19 6c"
ethernet0.dvs.portId = "129"
ethernet0.dvs.portgroupId = "dvportgroup-187"
ethernet0.dvs.connectionId = "1353964179"
```

ethernet0.addressType = "vpx"

ethernet0.generatedAddress = "00:50:56:ab:2e:40"

ethernet1.present = "true"

ethernet1.dvs.switchId = "73 40 2b 50 c6 ff f6 e0-fc 81 34 cb 42 63 19 6c"

ethernet1.dvs.portId = "102"

ethernet1.dvs.portgroupId = "dvportgroup-186"

ethernet1.dvs.connectionId = "1353979804"

ethernet1.addressType = "vpx"

ethernet1.generatedAddress = "00:50:56:ab:1e:7a"

tools.syncTime = "TRUE"

guestOS = "centos"

uuid.bios = "42 2b 39 1f 45 2b d0 3f-5b e3 e9 9c 47 9a c0 c1"

vc.uuid = "50 2b 11 4f ff 21 05 ef-bc a6 b1 9c 6e 96 76 5c"

snapshot.action = "keep"

sched.cpu.min = "0"

sched.cpu.units = "mhz"

sched.cpu.shares = "normal"

sched.mem.minsize = "0"

sched.mem.shares = "normal"

tools.upgrade.policy = "upgradeAtPowerCycle"

replay.supported = "FALSE"

debugStub.linuxOffsets =

"0x0,0xffffffff,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0"

replay.filename = ""

scsi0:0.redo = ""

pciBridge0.pciSlotNumber = "17"

pciBridge4.pciSlotNumber = "21"

pciBridge5.pciSlotNumber = "22"

pciBridge6.pciSlotNumber = "23"

pciBridge7.pciSlotNumber = "24"

scsi0.pciSlotNumber = "16"

ethernet0.pciSlotNumber = "32"

vmci0.pciSlotNumber = "33"

vmotion.checkpointFBSize = "4194304"

hostCPUID.0 = "0000000a756e65476c65746e49656e69"

hostCPUID.1 = "000006f2000208000000e3bdbfebfbff"

hostCPUID.80000001 = "00000000000000000000000120000800"

guestCPUID.0 = "0000000a756e65476c65746e49656e69"

guestCPUID.1 = "000006f200010800800022010febfbff"

guestCPUID.80000001 = "00000000000000000000000120000800"

userCPUID.0 = "0000000a756e65476c65746e49656e69"

userCPUID.1 = "000006f2000208000000e3bdbfebfbff"

userCPUID.80000001 = "00000000000000000000000120000800"

evcCompatibilityMode = "FALSE"

tools.remindInstall = "true"

ethernet1.features = "1"

ethernet1.pciSlotNumber = "34"

vmci0.id = "1201324225"

uuid.location = "56 4d 4e 9a 09 0c e4 6c-c8 cd ab cf a0 dd 7f e0"

cleanShutdown = "FALSE"

sched.swap.derivedName = "/vmfs/volumes/4f58d85a-cf6241e0-77de-

0019b9348420/attack/CentOS 2-4747535c.vswp"

CentOS VMX Configuration File

```
.encoding = "UTF-8"
config.version = "8"
virtualHW.version = "7"
pciBridge0.present = "true"
pciBridge4.present = "true"
pciBridge4.virtualDev = "pcieRootPort"
pciBridge4.functions = "8"
pciBridge5.present = "true"
pciBridge5.virtualDev = "pcieRootPort"
pciBridge5.functions = "8"
pciBridge6.present = "true"
pciBridge6.virtualDev = "pcieRootPort"
pciBridge6.functions = "8"
pciBridge7.present = "true"
pciBridge7.virtualDev = "pcieRootPort"
pciBridge7.functions = "8"
vmci0.present = "true"
nvram = "CentOS Host 1.nvram"
virtualHW.productCompatibility = "hosted"
powerType.powerOff = "soft"
powerType.powerOn = "hard"
powerType.suspend = "hard"
```

powerType.reset = "soft"

displayName = "Cent Host 1"

extendedConfigFile = "CentOS Host 1.vmxf"

floppy0.present = "true"

scsi0.present = "true"

scsi0.sharedBus = "none"

scsi0.virtualDev = "lsilogic"

memsize = "256"

scsi0:0.present = "true"

scsi0:0.fileName = "CentOS Host 1.vmdk"

scsi0:0.deviceType = "scsi-hardDisk"

sched.scsi0:0.shares = "normal"

sched.scsi0:0.throughputCap = "off"

ide1:0.present = "true"

ide1:0.deviceType = "cdrom-image"

floppy0.startConnected = "false"

floppy0.fileName = ""

floppy0.clientDevice = "true"

ethernet0.present = "true"

ethernet0.dvs.switchId = "73 40 2b 50 c6 ff f6 e0-fc 81 34 cb 42 63 19 6c"

ethernet0.dvs.portId = "131"

ethernet0.dvs.portgroupId = "dvportgroup-187"

ethernet0.dvs.connectionId = "1870144902"

ethernet0.addressType = "vpx"

ethernet0.generatedAddress = "00:50:56:ab:2e:2a"

tools.syncTime = "true"

guestOS = "centos"

uuid.bios = "42 2b e7 19 52 fa 5d ab-cf ae cd a5 d5 a9 13 9b"

vc.uuid = "50 2b bc 79 71 43 2c 9c-4a 8e 36 8b c7 92 b1 e3"

snapshot.action = "keep"

sched.cpu.min = "0"

sched.cpu.units = "mhz"

sched.cpu.shares = "normal"

sched.mem.minsize = "0"

sched.mem.shares = "normal"

tools.upgrade.policy = "upgradeAtPowerCycle"

replay.supported = "FALSE"

debugStub.linuxOffsets =

"0x0,0xffffffff,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0"

replay.filename = ""

scsi0:0.redo = ""

pciBridge0.pciSlotNumber = "17"

pciBridge4.pciSlotNumber = "21"

pciBridge5.pciSlotNumber = "22"

pciBridge6.pciSlotNumber = "23"

pciBridge7.pciSlotNumber = "24"

scsi0.pciSlotNumber = "16"

ethernet0.pciSlotNumber = "32"

vmci0.pciSlotNumber = "33"

vmotion.checkpointFBSize = "4194304"

hostCPUID.0 = "0000000a756e65476c65746e49656e69"

hostCPUID.1 = "000006fb000208000000e3fdbfebfbff"

hostCPUID.80000001 = "00000000000000000000000120100800"

guestCPUID.0 = "0000000a756e65476c65746e49656e69"

guestCPUID.1 = "000006fb00010800800022010febfbff"

guestCPUID.80000001 = "00000000000000000000000120100800"

userCPUID.0 = "0000000a756e65476c65746e49656e69"

userCPUID.1 = "000006fb000208000000e3fdbfebfbff"

userCPUID.80000001 = "00000000000000000000000120100800"

evcCompatibilityMode = "FALSE"

tools.remindInstall = "TRUE"

vmci0.id = "-710339685"

uuid.location = "56 4d 79 bf fa 4e 36 12-34 d9 7c 18 ac c0 51 80"

cleanShutdown = "TRUE"

sched.swap.derivedName = "/vmfs/volumes/4da486d8-e905eb60-42de-

001e4fc7d51f/CentOS Host 1 (101)/CentOS Host 1-db6c189d.vswp"

CentOS vSniffer VMX Configuration File

.encoding = "UTF-8"

config.version = "8"

virtualHW.version = "7"

pciBridge0.present = "true"

pciBridge4.present = "true"

pciBridge4.virtualDev = "pcieRootPort"

pciBridge4.functions = "8"

pciBridge5.present = "true"

pciBridge5.virtualDev = "pcieRootPort"

pciBridge5.functions = "8"

pciBridge6.present = "true"

pciBridge6.virtualDev = "pcieRootPort"

pciBridge6.functions = "8"

pciBridge7.present = "true"

pciBridge7.virtualDev = "pcieRootPort"

pciBridge7.functions = "8"

vmci0.present = "true"

nvram = "CentOS Host 1.nvram"

virtualHW.productCompatibility = "hosted"

powerType.powerOff = "soft"

powerType.powerOn = "hard"

powerType.suspend = "hard"

powerType.reset = "soft"

displayName = "Cent Host 1"

extendedConfigFile = "CentOS Host 1.vmxf"

floppy0.present = "true"

scsi0.present = "true"

scsi0.sharedBus = "none"

scsi0.virtualDev = "lsilogic"

memsize = "256"

scsi0:0.present = "true"

scsi0:0.fileName = "CentOS Host 1.vmdk"

scsi0:0.deviceType = "scsi-hardDisk"

sched.scsi0:0.shares = "normal"

sched.scsi0:0.throughputCap = "off"

ide1:0.present = "true"

ide1:0.deviceType = "cdrom-image"

floppy0.startConnected = "false"

floppy0.fileName = ""

floppy0.clientDevice = "true"

ethernet0.present = "true"

ethernet0.dvs.switchId = "73 40 2b 50 c6 ff f6 e0-fc 81 34 cb 42 63 19 6c"

ethernet0.dvs.portId = "131"

ethernet0.dvs.portgroupId = "dvportgroup-187"

ethernet0.dvs.connectionId = "1870144902"

ethernet0.addressType = "vpx"

ethernet0.generatedAddress = "00:50:56:ab:2e:2a"

tools.syncTime = "true"

guestOS = "centos"

uuid.bios = "42 2b e7 19 52 fa 5d ab-cf ae cd a5 d5 a9 13 9b"

vc.uuid = "50 2b bc 79 71 43 2c 9c-4a 8e 36 8b c7 92 b1 e3"

snapshot.action = "keep"

sched.cpu.min = "0"

sched.cpu.units = "mhz"

sched.cpu.shares = "normal"

sched.mem.minsize = "0"

sched.mem.shares = "normal"

tools.upgrade.policy = "upgradeAtPowerCycle"

replay.supported = "FALSE"

debugStub.linuxOffsets =

"0x0,0xffffffff,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0"

replay.filename = ""

scsi0:0.redo = ""

pciBridge0.pciSlotNumber = "17"

pciBridge4.pciSlotNumber = "21"

pciBridge5.pciSlotNumber = "22"

pciBridge6.pciSlotNumber = "23"

pciBridge7.pciSlotNumber = "24"

scsi0.pciSlotNumber = "16"

ethernet0.pciSlotNumber = "32"

vmci0.pciSlotNumber = "33"

vmotion.checkpointFBSize = "4194304"

hostCPUID.0 = "0000000a756e65476c65746e49656e69"

hostCPUID.1 = "000006fb000208000000e3fdbfebfbff"

hostCPUID.80000001 = "00000000000000000000000120100800"

guestCPUID.0 = "0000000a756e65476c65746e49656e69"

guestCPUID.1 = "000006fb00010800800022010febfbff"

guestCPUID.80000001 = "00000000000000000000000120100800"

userCPUID.0 = "0000000a756e65476c65746e49656e69"

userCPUID.1 = "000006fb000208000000e3fdbfebfbff"

userCPUID.80000001 = "00000000000000000000000120100800"

evcCompatibilityMode = "FALSE"

tools.remindInstall = "TRUE"

vmci0.id = "-710339685"

uuid.location = "56 4d 79 bf fa 4e 36 12-34 d9 7c 18 ac c0 51 80"

cleanShutdown = "TRUE"

sched.swap.derivedName = "/vmfs/volumes/4da486d8-e905eb60-42de-

001e4fc7d51f/CentOS Host 1 (101)/CentOS Host 1-db6c189d.vswp"

Appendix D Virtual Machine MAC Addresses

Table D.1.

MAC Addresses Used by the Virtual Machines

Virtual Machine	Interface	MAC Address
Attack1	eth0	00:50:56:AB:2E:40
Attack1	eth1	00:50:56:AB:1E:7A
Attack2	eth1	00:50:56:AB:2E:47
Attack3	eth2	00:50:56:AB:56:3B
Attack4	eth0	00:50:56:AB:1E:B7
Attack4	eth1	00:50:56:AB:1E:B8
CentOS1	eth0	00:50:56:AB:2E:2A
CentOS2	eth0	00:50:56:AB:56:3C
CentOS3	eth0	00:50:56:AB:1E:7F
CentOS4	eth0	00:50:56:AB:1E:A1
vSniffer1	eth0	00:50:56:AB:56:3E
vSniffer1	eth1	00:50:56:AB:56:3F
vSniffer2	eth0	00:50:56:AB:1E:7B
vSniffer2	eth1	00:50:56:AB:1E:7C

Table D.1. Continued

vSniffer3	eth0	00:50:56:AB:1E:7D
vSniffer3	eth1	00:50:56:AB:1E:7E
vSniffer4	eth0	00:50:56:AB:1E:B5
vSniffer4	eth1	00:50:56:AB:1E:B6

Appendix E Private VLAN VSM Configuration

```
version 4.2(1)SV1(5.1)
no feature telnet
feature private-vlan

username admin password 5 $1$EyzSqiwo$EE7qs0xQTT80dpVWOlhaZ.  role network-
admin
username ben password 5 $1$P3nTU.oq$EttRTOP/zFqOxSvDskr3y.  role network-
operator

banner motd #Nexus 1000v Switch#

ip domain-lookup
hostname Nexus100V
system default switchport
logging event link-status default
vem 3
  host vmware id 44454c4c-5300-1043-8036-b9c04f304731
vem 4
  host vmware id 44454c4c-4e00-1037-8044-b7c04f574331
vem 5
```

```
  host vmware id 44454c4c-5600-1038-8044-c7c04f574331
snmp-server user ben network-operator auth md5
0x1800e04e7dcfa9c4906ed37a9659fb30 priv 0x1800e04e7dcfa9c4906ed37a9659fb30
localizedkey
snmp-server user admin network-admin auth md5
0x1800e04e7dcfa9c4906ed37a9659fb30 priv 0x1800e04e7dcfa9c4906ed37a9659fb30
localizedkey

vrf context management
  ip route 0.0.0.0/0 10.19.35.1
vlan 1,971-974,1935
vlan 1

vlan 971
  name Control
vlan 972
  name Packet
vlan 973
  name SecureVLAN
  private-vlan isolated
vlan 974
  name PrivateVLAN
  private-vlan primary
```

```
  private-vlan association 973
vlan 1935
  name Management


port-channel load-balance ethernet source-mac
port-profile default max-ports 32
port-profile type ethernet Unused_Or_Quarantine_Uplink
  vmware port-group
  shutdown
  description Port-group created for Nexus1000V internal usage. Do not use.
  state enabled
port-profile type vethernet Unused_Or_Quarantine_Veth
  vmware port-group
  shutdown
  description Port-group created for Nexus1000V internal usage. Do not use.
  state enabled
port-profile type vethernet n1kv-system-control
  vmware port-group
  switchport mode access
  switchport access vlan 971
  no shutdown
  system vlan 971
  state enabled
```

```
port-profile type vethernet n1kv-system-management
  vmware port-group
  switchport mode access
  switchport access vlan 1935
  no shutdown
  system vlan 1935
  state enabled
port-profile type vethernet n1kv-system-packet
  vmware port-group
  switchport mode access
  switchport access vlan 972
  no shutdown
  system vlan 972
  state enabled
port-profile type ethernet n1kv-uplink0
  vmware port-group
  switchport mode private-vlan trunk promiscuous
  switchport trunk allowed vlan 1,971-974,1935
  switchport private-vlan trunk allowed vlan 1,971-974,1935
  switchport private-vlan mapping trunk 974 973
  channel-group auto mode on mac-pinning
  no shutdown
  system vlan 971-974,1935
```

```
  state enabled
port-profile type vethernet secureVlan
  vmware port-group
  switchport access vlan 973
  switchport mode private-vlan host
  switchport private-vlan host-association 974 973
  no shutdown
  description second data vlan
  state enabled
port-profile type vethernet ProtectedVLAN
  vmware port-group
  switchport mode access
  switchport access vlan 1
  switchport trunk native vlan 1
  no shutdown
  description Protected data vlan
  state enabled
port-profile type vethernet VirtualSniffer
  vmware port-group
  switchport mode access
  no shutdown
  description Virtual Sniffer
  state enabled
```

```
system storage-loss log time 30
vdc Nexus100V id 1
  limit-resource vlan minimum 16 maximum 2049
  limit-resource monitor-session minimum 0 maximum 2
  limit-resource vrf minimum 16 maximum 8192
  limit-resource port-channel minimum 0 maximum 768
  limit-resource u4route-mem minimum 1 maximum 1
  limit-resource u6route-mem minimum 1 maximum 1
  limit-resource m4route-mem minimum 58 maximum 58
  limit-resource m6route-mem minimum 8 maximum 8



interface port-channel1
  inherit port-profile n1kv-uplink0
  vem 3

interface port-channel2
  inherit port-profile n1kv-uplink0
  vem 4

interface port-channel3
  inherit port-profile n1kv-uplink0
  vem 5
```

```
interface mgmt0
  ip address 10.19.35.99/24

interface Vethernet1
  inherit port-profile n1kv-system-control
  description Nexus1000v, Network Adapter 1
  vmware dvport 64 dvswitch uuid "73 40 2b 50 c6 ff f6 e0-fc 81 34 cb 42 63 19 6c"
  vmware vm mac 0050.56AB.2E44

interface Vethernet2
  inherit port-profile n1kv-system-management
  description Nexus1000v, Network Adapter 2
  vmware dvport 100 dvswitch uuid "73 40 2b 50 c6 ff f6 e0-fc 81 34 cb 42 63 19 6c"
  vmware vm mac 0050.56AB.2E45

interface Vethernet3
  inherit port-profile n1kv-system-packet
  description Nexus1000v, Network Adapter 3
  vmware dvport 128 dvswitch uuid "73 40 2b 50 c6 ff f6 e0-fc 81 34 cb 42 63 19 6c"
  vmware vm mac 0050.56AB.2E46

interface Vethernet4
  inherit port-profile n1kv-system-management
```

```
  description VMware VMkernel, vmk0
  vmware dvport 101 dvswitch uuid "73 40 2b 50 c6 ff f6 e0-fc 81 34 cb 42 63 19 6c"
  vmware vm mac 0019.B934.8420

interface Vethernet5
  inherit port-profile n1kv-system-management
  description VMware VMkernel, vmk0
  vmware dvport 102 dvswitch uuid "73 40 2b 50 c6 ff f6 e0-fc 81 34 cb 42 63 19 6c"
  vmware vm mac 0019.B932.BF94

interface Vethernet6
  inherit port-profile n1kv-system-packet
  description CentOS Host 1 (101), Network Adapter 1
  vmware dvport 131 dvswitch uuid "73 40 2b 50 c6 ff f6 e0-fc 81 34 cb 42 63 19 6c"
  vmware vm mac 0050.56AB.2E2A

interface Vethernet7
  inherit port-profile n1kv-system-packet
  description BackTrack, Network Adapter 1
  vmware dvport 129 dvswitch uuid "73 40 2b 50 c6 ff f6 e0-fc 81 34 cb 42 63 19 6c"
  vmware vm mac 0050.56AB.2E47

interface Vethernet8
```

```
  inherit port-profile secureVlan
  description CentOS Host 2, Network Adapter 1
  vmware dvport 480 dvswitch uuid "73 40 2b 50 c6 ff f6 e0-fc 81 34 cb 42 63 19 6c"
  vmware vm mac 0050.56AB.563C

interface Vethernet9
  inherit port-profile n1kv-system-packet
  description BackTrack2, Network Adapter 1
  vmware dvport 132 dvswitch uuid "73 40 2b 50 c6 ff f6 e0-fc 81 34 cb 42 63 19 6c"
  vmware vm mac 0050.56AB.563B

interface Vethernet10
  inherit port-profile secureVlan
  description attack, Network Adapter 1
  vmware dvport 482 dvswitch uuid "73 40 2b 50 c6 ff f6 e0-fc 81 34 cb 42 63 19 6c"
  vmware vm mac 0050.56AB.2E40

interface Vethernet11
  inherit port-profile n1kv-system-packet
  description Virtual Sniffer 10.19.35.197, Network Adapter 1
  vmware dvport 135 dvswitch uuid "73 40 2b 50 c6 ff f6 e0-fc 81 34 cb 42 63 19 6c"
  vmware vm mac 0050.56AB.563E
```

```
interface Vethernet12
  inherit port-profile n1kv-system-management
  description Virtual Sniffe...19.35.197, Network Adapter 2
  vmware dvport 103 dvswitch uuid "73 40 2b 50 c6 ff f6 e0-fc 81 34 cb 42 63 19 6c"
  vmware vm mac 0050.56AB.563F

interface Vethernet13
  inherit port-profile n1kv-system-management
  description attack, Network Adapter 2
  vmware dvport 104 dvswitch uuid "73 40 2b 50 c6 ff f6 e0-fc 81 34 cb 42 63 19 6c"
  vmware vm mac 0050.56AB.1E7A

interface Vethernet14
  inherit port-profile VirtualSniffer
  description Virtual Sniffer 2, Network Adapter 1
  vmware dvport 640 dvswitch uuid "73 40 2b 50 c6 ff f6 e0-fc 81 34 cb 42 63 19 6c"
  vmware vm mac 0050.56AB.1E7B

interface Vethernet15
  inherit port-profile n1kv-system-management
  description Virtual Sniffer 2, Network Adapter 2
  vmware dvport 105 dvswitch uuid "73 40 2b 50 c6 ff f6 e0-fc 81 34 cb 42 63 19 6c"
  vmware vm mac 0050.56AB.1E7C
```

```
interface Vethernet16
  inherit port-profile n1kv-system-packet
  description Virtual Sniffer 3, Network Adapter 1
  vmware dvport 137 dvswitch uuid "73 40 2b 50 c6 ff f6 e0-fc 81 34 cb 42 63 19 6c"
  vmware vm mac 0050.56AB.1E7D

interface Vethernet17
  inherit port-profile n1kv-system-management
  description Virtual Sniffer 3, Network Adapter 2
  vmware dvport 106 dvswitch uuid "73 40 2b 50 c6 ff f6 e0-fc 81 34 cb 42 63 19 6c"
  vmware vm mac 0050.56AB.1E7E

interface Vethernet18
  inherit port-profile secureVlan
  description CentOS Host 3, Network Adapter 1
  vmware dvport 481 dvswitch uuid "73 40 2b 50 c6 ff f6 e0-fc 81 34 cb 42 63 19 6c"
  vmware vm mac 0050.56AB.1E7F

interface Ethernet3/2
  inherit port-profile n1kv-uplink0

interface Ethernet4/1
  inherit port-profile n1kv-uplink0
```

```
interface Ethernet5/1
  inherit port-profile n1kv-uplink0


interface control0
line console
boot kickstart bootflash:/nexus-1000v-kickstart-mz.4.2.1.SV1.5.1.bin sup-1
boot system bootflash:/nexus-1000v-mz.4.2.1.SV1.5.1.bin sup-1
boot kickstart bootflash:/nexus-1000v-kickstart-mz.4.2.1.SV1.5.1.bin sup-2
boot system bootflash:/nexus-1000v-mz.4.2.1.SV1.5.1.bin sup-2
monitor session 1
  source vlan 973-974 both
  destination interface Vethernet14
  no shut
monitor session 2
  no shut
svs-domain
  domain id 555
  control vlan 971
  packet vlan 972
  svs mode L2
svs connection vcenter
  protocol vmware-vim
  remote ip address 10.19.35.50 port 80
```

```
  vmware dvs uuid "73 40 2b 50 c6 ff f6 e0-fc 81 34 cb 42 63 19 6c" datacenter-name Lab
  max-ports 8192
  connect
vsn type vsg global
  tcp state-checks
vnm-policy-agent
  registration-ip 0.0.0.0
  shared-secret **********
  log-level
```