Toward An Active Network Security Architecture

by

Ryan S. Hand

Bachelor of Science, Information Technology, 2006,

George Mason University, Fairfax, Virginia

Master of Science, Information Technology, 2011,

University of Maryland University College, Adelphi, Maryland

A thesis submitted to the

Faculty of the Graduate School of the

University of Colorado in partial fulfillment

of the requirements for the degree of

Master of Science

Department of Computer Science

2014

This thesis entitled:
Toward An Active Network Security Architecture
written by Ryan S. Hand
has been approved for the Department of Computer Science

_____

Dr. Eric Keller

_____

Dr. Douglas Sicker

_____

Dr. Shivakant Mishra

Date _____

The final copy of this thesis has been examined by the signatories, and we find that both the content and the form meet acceptable presentation standards of scholarly work in the above mentioned discipline.

Hand, Ryan S. (M.S., Computer Science)

Toward An Active Network Security Architecture

Thesis directed by Dr. Eric Keller

Network and systems security have never been more important than they are today. Attackers continue to expose new vulnerabilities and exploit them as quickly as new technologies, applications, and security strategies are developed. Today's security systems work in relative isolation with limited programmatic control and remediation is working at human reaction speed. Active Security gives a fundamental architectural advantage to the network defender.

Active Security seeks to leverage all resources present throughout the infrastructure through a unified programming interface to protect existing infrastructure, interface with a variety of sensors, adjust the configuration at run-time, collect forensic data on-demand, and counter an attack. This makes a programmable network infrastructure and Software-Defined Networking (SDN) control key enablers of an Active Security architecture.

Today, digital forensics is commonly performed in response to an incident or anomaly after an attacker has succeeded and possibly cleaned up the crime scene. We argue that this deep well of useful information should be leveraged in our architecture immediately when an attack or anomaly is detected. In particular, we investigate host physical memory which is often lost or tampered prior to an investigation.

SDN bears the banner of programmatic control and thrives in an infrastructure capable of granular programming. This requirement acts as a barrier to entry by enterprise organizations who neither have the funding nor the technical ability to upgrade their network to SDN control overnight. A new transition alternative is needed that allows SDN control today using the existing legacy equipment. We show significant progress at closing this gap with ClosedFlow, extending SDN control to legacy networks, to enable immediate adoption of an Active Security architecture in the enterprise.

We strengthen the case for an active security architecture illustrating the benefits of an automated sense, decide, respond feedback loop within a software-defined security controller and present the two research branches that were specifically investigated.

The work presented in this thesis supports a larger collaborative research effort. Active Security was presented last November at the 12th ACM Workshop on Hot Topics in Networks (HotNets). The supporting research on proprietary network programmability presented here is currently under submission.

Dedication

I wish to dedicate this work to my wife Stephanie and our four amazing children, Caleb, Baylee, Ainsley, and Kerrigan. Thank you for your love and support during this chapter of our lives.

Thank you to my parents Steven and Shelley Hand for always emphasizing the importance of continued, life-long education and the responsibility to our children that comes with having achieved it.

Acknowledgements

Dr. Eric Keller, thank you for leading and teaching me along my research journey from a semester project, to a valuable, long-term research effort. I appreciate the time you spent coaching me to a top-tier publication and workshop presentation. Most of all, thank you for your patience with my endless barrage of questions and for empowering, challenging, and giving me a true appreciation for the research process. I will carry this with me the rest of my professional and academic career.

Thank you to Dr. Doug Sicker for taking a chance on a young Army Officer who aspired to be a computer scientist. I truly appreciate your guidance and direction throughout my time in the program and helping me to get the right courses at the exact right time to create an academic experience like few others. Truly exceptional.

Dr. Shivakant Mishra, one of my only regrets throughout this program is that I won't be here long enough to attend another operating systems course of yours! Your teaching method, in-depth assignments, and advice throughout my course project inspired me to dig deeper into the area of Distributed Systems and consider its implications in systems and application development. Many thanks.

I would also like to thank Professor Jose Santos for his support and expert advice throughout my time at the University of Colorado on multiple projects and problem sets. Thank you for allowing me to make the Telecommunications Lab a second home!

Contents

Chapter

Figures

Figure

Chapter 1

Introduction

Active Security is a new methodology that provides a centralized programming interface to control the detection of attacks, data collection for attack attribution, configuration revision, and reaction to attacks. Rather than settle for a system which has many individual components which can capture, detect, and even block traffic, having programmatic control would, for example, allow for one component to detect something (having partial information), trigger an automated response to investigate, perhaps alter the network based on the findings, and reconfigure security devices with an updated view of threats. To realize Active Security, a centralized, active security controller interfaces with various sensors, network equipment, host, and security systems allowing for programmatic control over an event driven feedback loop of the entire cycle of configuration, detection, investigation, and response.

Active Security is primarily motivated by two high level problems with our modern defenses: Security systems are working in relative isolation and complex remedial actions to the network are often based on human reaction speeds.

1.1 Problem with Security Today

We argue that todays security mechanisms are too rigid and that a highly dynamic and programmable security infrastructure is needed. Here we motivate with discussion of todays security systems and provide an example of where this rigidity limits the capabilities of the security system.

- **Firewalls**: (e.g. Cisco ASA, Pfsense, Sophos UTM) can block traffic based on the spec-

ified configuration and a set of known signatures. They are good at blocking traffic that must pass through them, but attackers often find holes in the firewall configuration and if ingress and egress traffic rules are misconfigured, a client-side attack could easily establish a foothold from within and communicate out. Perimeter firewalls also do little to mitigate propagation of an attack laterally between network segments allowing them to pivot and attempt to escalate their privilege level.

- **Intrusion Detection/Prevention Systems**: are capable of monitoring traffic (e.g. Sophos, Snort), but require many machines to handle a full traffic load. As such, traffic is often sampled, leading to only examining a sub-set of traffic. Explicit traffic signatures must be specified in traditional configurations, with databases of known signatures growing daily. Newer techniques such as those that use learning to detect anomalies are a step in the right direction, but still only have a limited vantage point on which to base the judgment. Intrusion detection systems can only alert problems, and intrusion prevention systems can block traffic, but cannot perform more complex actions on their own.

- **Digital Forensics and Incident Response**: as we will later describe, digital forensics and incident response are vast and incredibly diverse fields of study. Each new piece of hardware and software brings new context and artifacts to an investigation. Digital forensics involves not only tools (e.g. FTK, Volatility, EnCase) but techniques to develop a larger picture of what happened during an attack, crime, incident, etc. The rich information involved is often analyzed hours, days, or even months after an event has occurred.

- **Security Information and Event Management**: provide real-time aggregation of events and logged information (e.g. HP Arcsight Express, McAfee Enterprise Security Manager, Solarwinds) that can be incredibly difficult for human eyes to discern and sift through. These are a step in the right direction, but still limited from a programmability standpoint.

- **Policy Models**: Security models, like defense-in-depth, provide a guide for security and executive leadership to follow, but fail to fill the gaps between each of the above listed security and monitoring systems. We force our protection hardware to work in relative isolation and often fail to fully utilize existing capabilities or to include built-in functionality for robust interaction and programmability.

In each case, these only have limited response measures (e.g., updating the firewall to block traffic) and have static configurations that require human intervention to change any aspect of the configuration (such as altering whats being monitored or altering the IP address assignment to protect a current target).

Academic research efforts such as FRESCO [56] illustrate the desire to leverage the granularity of control that SDN affords to rapidly create and deploy security functionality (firewalls, deflectors, scan detection, and IDS). Additionally, AVANT-GUARD [57] uses statistics and counter information from OpenFlow switches to define anomalous conditions that trigger actions in the network.

Despite the quality research efforts that have been made in the field of security with respect to SDN, we believe that we need to go even further to monitor, interface with, configure, and control the entire infrastructure. The power lies in context aware automation proposed with active security.

1.2 The Need for a Programmatic Feedback Loop

Our Active Security concept is partly inspired by the OODA feedback loop introduced in 1976 by US Air Force Officer and renowned military strategist Col. John Boyd. OODA stands for:

- **Observe**: where we collect data and information based on input from our senses and information fed from events currently unfolding.

- **Orient**: where we analyze and synthesize the data to form perspective or context.

- **Decide**: based on the data and context, we form a hypothesis and choose a course of action.

- **Act**: we then physically implement the course of action.

And ultimately we reenter this cycle, constantly assessing and reassessing our situation and our surroundings as they unfold. These principles can and have been applied to many areas of military strategy and business. Boyd's concept even inspired research projects that produced the lighter weight, more maneuverable fighter jets of our time including the F-16 and F/A18, still in service today with the primary goal of giving our pilots the ability to react and make decisions quicker than the enemy can. In this way, we force the adversary to react to an environment that we control.

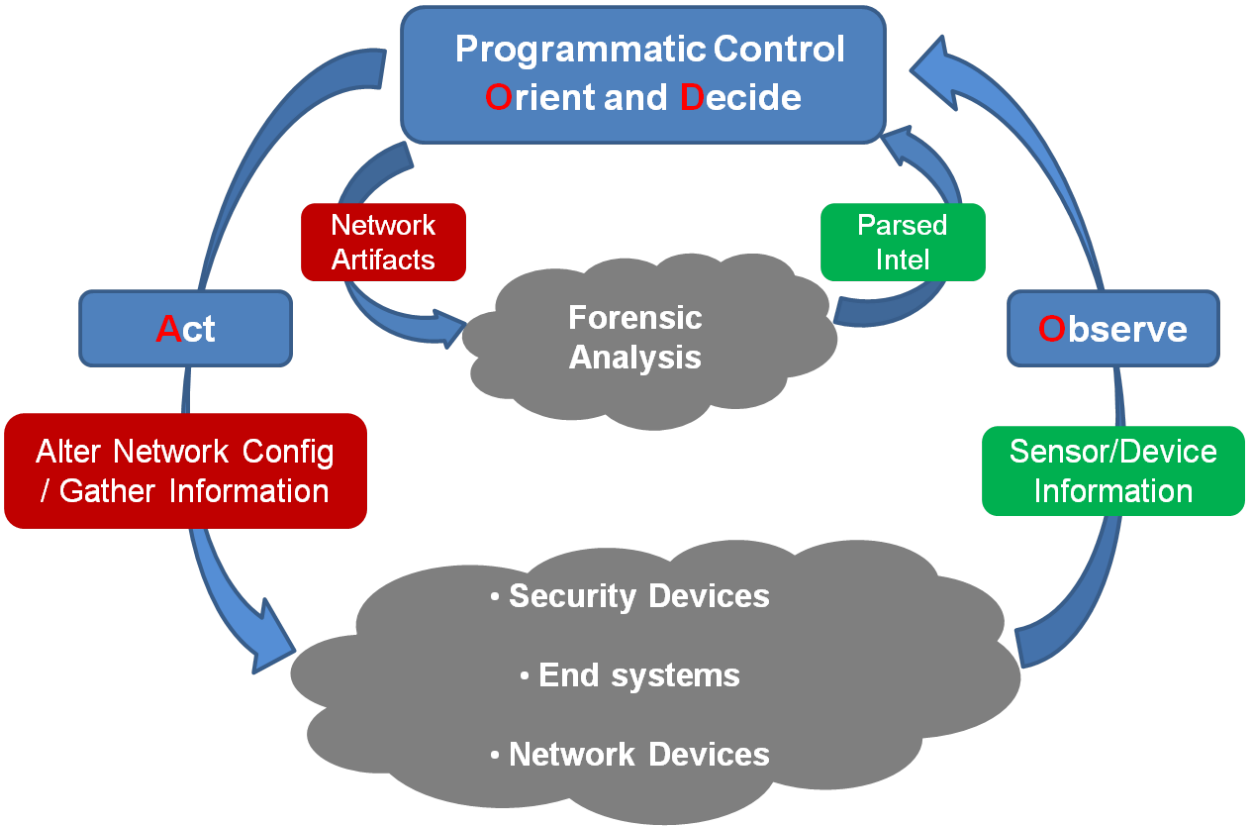
Illustrated in Figure 1.1, the goal of active security is to realize each of these 4 concepts through network sensors, device configuration, SDN control and decision logic, and automated programmatic actions based on the sensed information and situational context.

In industry, there have been point solutions to use a feedback loop to, for example, detect DDoS attack and reconfigure routers to block associated traffic [2]. The Cisco CSMARS [17] and SolarWinds [9] systems went a bit further and integrated some remediation within a SIEM product. These responses (or actions) were built-in and limited to capabilities of routers, as opposed to leveraging software-defined networking for more general actions and an extensible framework. They also limit the actions to defensive actions, rather than also incorporate information gathering actions as with active security. With active security we look to expand to a more general, open, and programmable solution which uses a continuous process of sensing and adaptation to discover and defend against threats, including those that have not been seen before.

1.3 Components of An Active Security Architecture

Active security couples passive components which monitor the state of the network and act according to some configuration, with highly dynamic components that enforce policy or manipulate traffic, and a programming environment to exercise granular control over each of these. At a high-

Figure 1.1: Applying the OODA concept, we see this as a continued cycle of observing sensory and configuration data from a multitude of device messages, traffic of interest, and alerts.
The intelligence lies in the controller where our ORIENT and DECIDE phases of the OODA loop occur. We may decide to gather further information from adjacent sources to get a better perspective, or we may decide alter the network configuration based on a flaw we have discovered. Information gathered may also be subjected to forensic analysis where we take network artifacts and seek to obtain parsed intelligence to better ORIENT the controller to the current network state and aid decision making to better protect the network or attribute an attack.



level, we are also inspired by concepts from software-defined networking in providing a centralized programmatic control over devices in the infrastructure.

Active security is centered around five core capabilities: protect, sense, adjust, collect, and counter, illustrated in the form of controller modules, similar to the evolution of device drivers, in Figure 1.2.

1.3.1 Protect: Configure the Infrastructure

As a first step, any infrastructure must be setup to be able to operate properly. Part of this is utilizing protection mechanisms which provide some level of security against common attack scenarios (using and configuring a firewall). Unfortunately, while configuration to protect an infrastructure is somewhat well understood today, it still leads to many errors [60]. Research has been performed in this space in how to realize less error-prone firewall configuration [20] [23]. This part of active security is not new, but it is an essential foundation upon which we may build a context-aware security system.

1.3.2 Sense: Interface To Many Different Sensors

The security controller must be able to accept information (alerts or other information) from a variety of sources that are performing some sort of detection and monitoring. Traditionally, intrusion detection systems (IDS) perform most of the detection as stand alone boxes, but there are many other potential sources of information e.g., an end-host firewall that receives a packet that it ends up blocking (but somehow made it past the IDS) may be able to notify the reactive system to pay attention to this kind of traffic, or an SDN network controller designed to provide security mechanisms within the network [56] may provide insight into activity within the network. Each of these is a sensor in an active security system. With active security, we can allow operators to program how different sensor inputs should be interpreted and combined in a context-aware manner to ultimately seek convergence upon a consistent configuration.

1.3.3 Adjust: Better Defense and Monitoring

A static network configuration gives attackers the ability to map out the infrastructure and plan an attack. The active security controller must therefore be able to control the configuration of the infrastructure (e.g., the assignment of IP addresses) to alter its behavior at run-time. Moving target defense [38] has been positioned as a counter measure, and even demonstrated within an OpenFlow network [37]. We believe that doing so in a targeted manner rather than randomly will be more effective. In order to achieve this, our active security controller can interface with a software-defined network controller [32] [46] to alter the network. A static network configuration also limits the visibility of what is happening on the network. It is not feasible to log all ingress and egress traffic, so instead, operators are forced to sample traffic or explicitly configure subsets of traffic to monitor (e.g., based on a central policy, Ethane could direct a sub-set of traffic through a proxy [26]). Instead of statically defining, we need the dynamic ability to alter what were looking at closely at run-time (in response to other information, such as alerts from firewalls or IDS). Going beyond simple re-direction of traffic, we can extend to more heavy-weight operations. Researchers have proposed mechanisms to rewind the network and replay it [61, 45]. Likewise, researchers have proposed mechanisms to ensure an 'accountable virtual machine' is behaving properly [33]. In each case, these actions, while useful, are expensive to perform. With active security, we have the ability to determine when these active actions should be run.

1.3.4 Collect: In-Attack Forensics

As part of the response to a potential attack that has been detected, we need to expose to the software control the ability to perform forensic evidence gathering in order to understand attacks and attribute it to an individual or organization. The challenge here is that there are many different sources of information with a wide diversity of devices (servers, routers, wireless access points, etc). As such, an extensible framework to plug-in evidence gathering mechanisms is needed. Importantly, this forensic evidence gathering needs to be performed in a stealthy and

non-intrusive manner. There have been a few research projects which indicate we will be able to perform the memory gathering in such a manner [22] [52]. We believe memory to be one of the more challenging sources of evidence to gather in a stealth manner, and believe other mechanisms will be possible. We discuss the value of collecting forensic artifacts from host physical memory in Chapter 2.

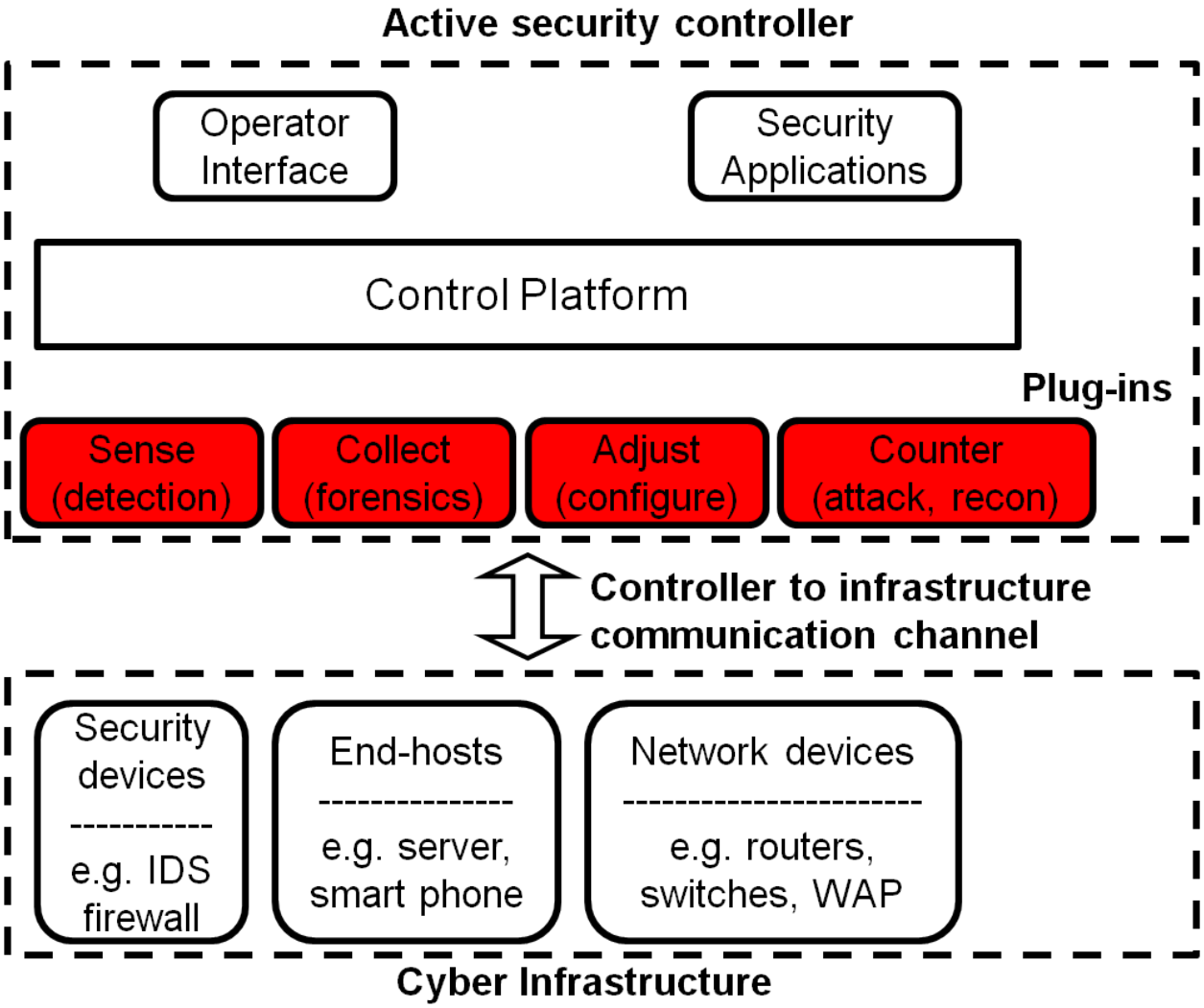
1.3.5 Counter: Reconnaissance and Offensive Actions

With an active security infrastructure in place, additional measures beyond protective measures become possible and attractive. In particular, we believe additional actions include reconnaissance and counter attack.

Reconnaissance: As an attack is detected, it may be valuable to allow the attack to continue, but monitor the activity closely. In doing so, we can learn a lot about the attacker such as identity, motive, and techniques. Honeypots have been used to attract attackers and studied their behavior. With active security, we look to go one step further and upon detection of a compromised system, effectively turn that system into one that closely monitors the activity giving the attacker the perception that they are interacting with a real system when, in reality, it is a system that records their actions, decisions, and reactions, giving insight into their methodology. In particular, we can migrate the malicious code and its open network connections to a dedicated virtual machine. With SDN providing control over the forwarding, we can then isolate any malicious traffic to the quarantined machine and all other traffic to the original victim machine, which continues to operate as normal after being cleaned up.

Counter Attack: In some cases, the best defense may be to attack the attacker (e.g., launching a denial of service may consume all of the attackers resources and limit their ability to continue an attack). We are providing the platform to be able to gather evidence, potentially during an attack. Once we gather this information, we can choose to counter the attack by going after the attacker.

Figure 1.2: Active Security controller architecture depicting modules to support an automated security feedback loop.



1.4 Research Supporting An Active Security Architecture

This thesis investigates actions beyond what is openly available today. In, particular we investigate two key areas that contribute to a complete Active Security architecture, and attempt to overcome specific limitations expressed in previous work in SDN and security.

First, we promote the use case of active, in-attack collection of forensic information for damage assessment, attribution, evidence preservation, and ultimately, to better protect the network utilizing it as input to an automated feedback cycle. Specifically, the collection of host system physical memory is examined, as this is an often lost or missing forensic artifact during incident response. In this way, active security seeks to eliminate, or significantly narrow, the gap between an attack and the subsequent incident response and remediation by preserving the often lost or corrupted system state for incident responders and forensic analysts.

Secondly, several quality research efforts have shown that there is virtually unlimited potential value having a programmable network with central control to alleviate management and security. Unfortunately, legacy enterprise hardware switches from many vendors are not compliant/compatible with protocols like OpenFlow. We argue that, to a large degree, this does not matter. We show that among three large vendors (Cisco, Juniper, and HP), we can exercise OpenFlow-like, programmatic control that enforces a centrally controlled policy and control logic over already existing multi-layer switch functionality. Our initial experiments target Cisco 3550 multi-layer switches and above. We wish to demonstrate that this "hardware gap" can be overcome to allow wider spread proliferation of software-defined networking technologies and further strengthen the case for the introduction of an Active Security architectural strategy for enterprise, datacenter, and ISP networks today.

Having now provided a background on the greater research effort in Active Security, we now focus more specifically on the value that live forensic collection can add to an architecture that implements an automated feedback loop.

Chapter 2

Proactive In-Attack Forensics

The field of digital forensics is an exciting one. With each new piece of technology, whether hardware or software, a new set of rich artifacts present themselves to analysts and investigators. In many cases, we see the same technologies applied to a new environment and can rely upon some forensic tools and techniques to remain the same but with slightly different context. We see often that forensic artifacts are harvested only after a serious incident or suspected intrusion has occurred. These actions are performed by humans aided by commercial software. A big problem is that not all investigators or analysts are equally talented or trained to understand the tools they use or the information they are gathering. This can mean the difference between a successful conviction or mistrial in court. In an Active Security architecture, we believe that these techniques can be defined programmatically and should leverage this rich information resource. An often lost artifact in investigations is non-volatile memory. We seek to collect this information at the time of anomaly detection and categorize them in two ways:

Information useful for damage assessment, attribution, and evidence: By the time digital forensics is performed, attackers have infiltrated the system, performed their malicious activity (e.g., extracted confidential data), and cleaned up the evidence trail (e.g., deleted log files). As an example, in enterprise networks, the importance of intellectual property is unmatched. Anyone from Fortune 500 companies like Coca-Cola or Apple, to credit card companies like Visa, or industrial (gas, power, etc) control systems are under attack on a daily basis. These attacks typically involve information gathering, active exfiltration of corporate data, and outright malicious

destruction of property and reputation. In almost all cases, the most important question asked by executive leadership and legal teams is, "what sensitive data was exposed"? If volatile memory is not captured during the attack, we may never know, as many exploits and their payloads can reside entirely in random access memory and have self-deleting behavior.

Information useful for better protection: Beyond simply providing evidence that can be used to attribute an attack and assess the damage, the information gathered during an attack can be useful to help defend against the on-going attack and help prevent future attacks. That is, this information can provide more input in the feedback loop. As an example, consider malware that has compromised a system and has begun to extract data. We may catch this through our intrusion detection system, and limit the damage. Suppose, however, the malware also contained code to spread itself within the local area network, which is unmonitored by the IDS. In this case, through in-attack forensic evidence gathering, we are able to detect extra network sockets that are open, and use this as input in the feedback loop, which will then trigger an action to reconfigure the network to block that traffic. The key here is again that we are able to perform information gather actions on demand in order to get the information during an attack.

2.1 Volatility: An Advanced Memory Forensics Framework

Volatility is an extremely powerful, and open, set of digital forensic tools used to dig out volatile memory artifacts. Volatility's power also comes from the variety of memory dump formats (32/64-bit, hibernation files, VM snapshots, etc) and platforms it supports (Windows, Linux, Mac, Android, ARM). Because memory dumps are parsed and analyzed from collected samples, we have the ability to creatively script our own collection templates and tools. There is also a strong level of community support that openly contribute scripts to detect advanced malware and rootkits. Below explained are several (but not limited to) of the powerful categories of tools that Volatility has to offer us.

- **Network Socket Information**: This feature is pretty straight forward as it displays for

us TCP and UDP endpoints and listeners. What is important to us, in many cases, is the system process ID that is associated with a particular network connection. This can lead us to investigate the process and its memory for further information. We can also gather the memory offset that the connection resides in and the socket state (i.e. listening, closed, etc.).

- **Process and DLL Information**: Volatility allows us to extract detailed information about running processes. We can list the active processes that were running and learn what SIDs, DLLs, handles (is it a file, registry key, process), PE version numbers, and environment variables were associated with each. This allows us to understand what privilege levels were being used by a process or could allow us to compare a PE file being used to a known good one. We also have utilities to extract command history, console information, and functions that were imported or exported by a process. We can also gain insights into problems based on what we don't see. For instance, if a process shows that it was spawned by a parent process that is not listed, this could be the result of a process being hidden from us or that a piece of malware migrated from the parent process and is seeking to become persistent in another, more stable process.

- **Windows Registry**: The Windows Registry is a wealth of information for a forensic analyst and often a place where attackers attempt to create persistent backdoors. Malicious registry key entries allow for instructions or programs of the attacker's choosing to run upon system boot or restart. Volatility allows us to examine hives, keys, subkeys and more.

- **Portable Executable Extraction**: We can also extract artifacts from process memory including a full working Portable Executable file that is associated with the process. This is an important piece of functionality as many executable files use obfuscation to defeat modern anti-virus scanners when stored on disk. However, when these files are run, they must decode themselves to perform their malicious action. Ultimately, we will find this decoded version of the file in physical memory, potentially including its encoding/encryption

scheme. Alternatively, we can dump the process to an executable memory sample for debugging function by function.

While each of these features provides detailed perspectives of the current running state of a system, they are only that, perspectives. This is why we have forensic analysts and investigators to give context to data to transform it into information. They scrutinize what may seem subtle at first, but one of these perspectives leads us to view memory from several other angles to develop the larger picture of how an attack happened, where it propagated, and what resources were compromised or stolen. In an active security architecture, this information could prove paramount to attributing an attack to a person, organization, or nation- state actor and allow us to better assess damage and remedy the issue, possibly before the damage is done. The key is to perform these actions right when a particular anomaly is detected.

2.2 Prototype

We built an initial prototype to test the feasibility of the core architecture. This initial prototype supports the motivating example of in-attack forensics. We envision a tight coupling with software-defined networking which provides the programmatic control over the network devices, and so we chose to integrate within the FloodLight [14] controller platform rather than run separately. We extended FloodLight to not only control the network, but to also interact with end-host systems and security middle-boxes.

Interaction with an IDS security middle-box: Within our test implementation, we chose the Snort [16] Intrusion Detection System as the source of sensory input to our active security controller. We parse the Snort IDS standard alert log with a middleware Perl script that continually parses the IDS log looking for alerts. When it discovers an intrusion alert, it obtains pertinent information from the log output, feeds it as input to the controller, and activates a controller module, in this case, a module designed to take a forensic image of volatile memory.

Interaction with an end-host forensic system: Upon receiving the trigger from the

IDS, our FloodLight module takes steps to load the Linux Memory Extractor (LiME) [15] Linux kernel module onto the infected end host, capture a dump of memory, send it back to the controller, and store it. The controller achieves this by establishing a secure connection with the victim host, remotely loading the kernel module, securely copying the resulting image back to the controller, and removing the kernel module following transfer of the file. Our controller module loads and unloads the LiME kernel module through an SSH call, and secure copies the resulting memory image back to the controller for analysis. A hash may be computed at a later time on the controller's copy of the image to verify its integrity, trustworthiness, and admissibility in court. SSH keys are used to authenticate the controller's actions on the victim host. The controller verifies the fingerprint of the victim's SSH key on each connection asserting the authenticity of the victim host. For our forensics example in the next section, we will examine an identical memory image obtained using WinPmem on a Windows platform to illustrate some of the Windows Registry functionality in Volatility.

Scripting with Volatility: Now that we have a forensic copy of volatile memory at the time of attack/anomaly detection, we can now utilize any tool we wish to parse through and dig out artifacts of interest. We introduced above the Volatility memory forensics framework, which gives us the ability to script our own situation specific tools. There are a grand number of other third-party utilities and tools that also offer this capability and could be leveraged by modules in an Active Security controller.

2.3 In-Attack Forensic Example

2.3.1 Scenario

We begin under the assumption that the network perimeter has already been breached indirectly through an unfiltered email attachment that reached a client system, seemingly from a trusted party. This type of client-side exploit became increasingly common prior to 2009, which included the RSA source code breach and Operation Aurora, and persists today.

The email attachment contains malicious payload that will attempt to establish a remote

control channel with an attacking host machine. In our implementation we evaded over 30 anti-virus products using a relatively simple code obfuscation technique allowing it to remain invisible on disk, but run correctly in memory. The payload is also scripted to establish a persistent backdoor that will "re-hook" the victim upon restart or user login by saving a script in a directory and calling it upon reboot using a Windows "RUN" registry key entry. Upon successful connection, these artifacts are commonly cleaned-up automatically. We want to capture this behavior before that is allowed to happen. To better protect our network and preserve the evidence for further post-postmortem analysis.

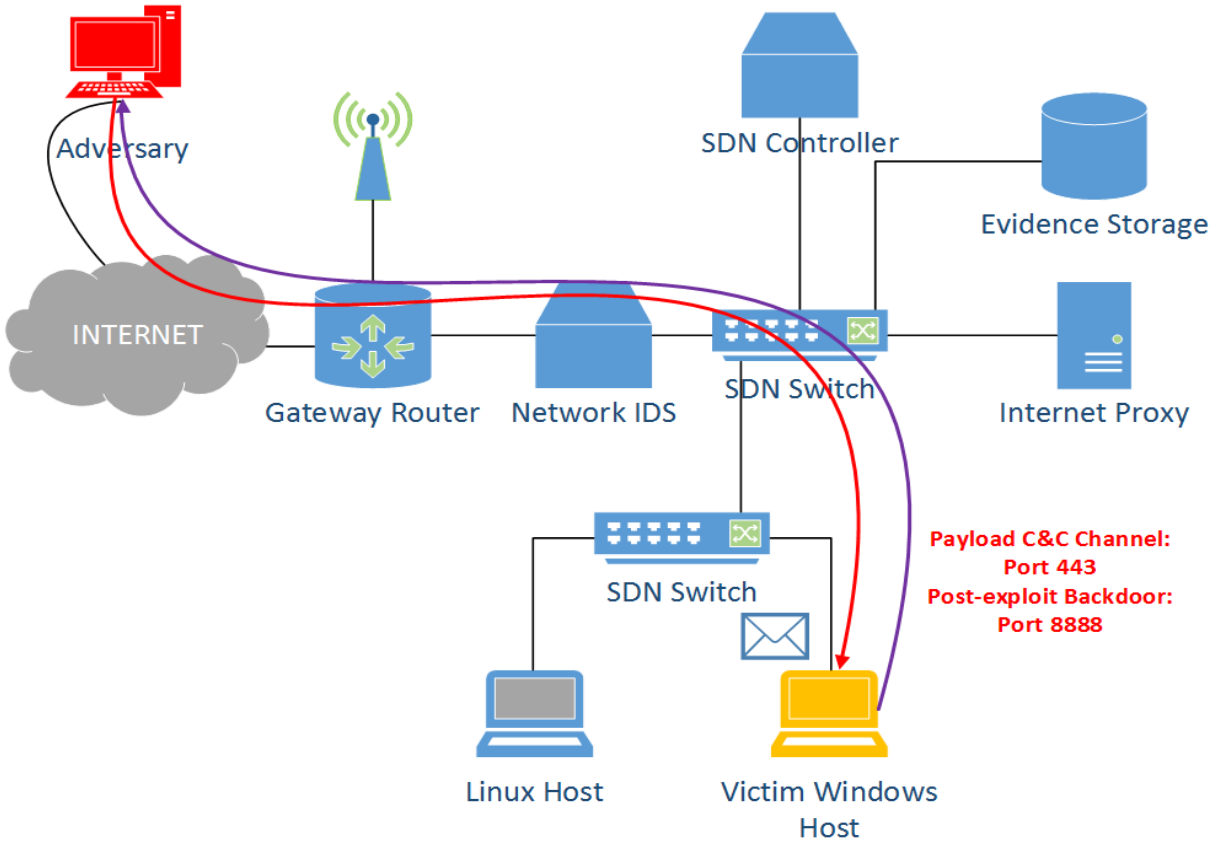
One final assumption is that our administrator is doing a fair job at inspecting egress traffic and the outbound connection over port 443 to the attacking host machine is detected by our IDS, beginning the triggering of our above explained prototype. We have now captured pertinent information about the source and destination IP addresses and port numbers of the systems involved and trigger the collection of volatile memory from the suspicious victim system. As a precaution, our network controller may direct action to route the victim system through an Internet proxy or halt connectivity altogether based on whether the source host is allowed https or if the destination IP is not included in a whitelist. We now discuss the automated forensic process over our victim host memory sample. We now illustrate a set of forensic actions that may be conducted in this particular scenario. Actions that may be conducted with Volatility are certainly not limited to these.

2.3.2 Captured Before Clean-up

Now that we have a forensically sound capture of the system state at the time of attack, we can preserve this image for incident responders if an investigation is deemed necessary. We can now also use this data to conduct our own examination.

Based on the information that the IDS provided us (e.g. offending source/destination IPs and ports), a logical first step in the forensic process is to verify the active, and even previous, network connections and their associated process identifiers (PIDs). We do this with Volatility's

Figure 2.1: This figure depicts our implementation environment for the attack scenario. The red directed arrow represents the attack vector which takes an indirect approach through a client-side exploit delivered via email attachment. The purple directed arrow represents command-and-control channel communication from the victim host back to the attacking host over port 443 for the initial exploit payload, and a second persistent network connection over port 8888, which installs itself as a Windows service and is called periodically until a connection is established.



"**connections**" utility shown in Figure 2.2. We can now see the PIDs associated with each of the connections in question. We may now further investigate into the origins of these processes. With

Figure 2.2: We first examine the suspicious network connections that our IDS detected. From these network connections, we have determined the offending processes from which they originated, 3376 and 4012 respectively.

```
Volatile Systems Volatility Framework 2.2
Offset(V)  Local Address             Remote Address            Pid
---------- ------------------------ ------------------------ ------
0x81df6640 192.168.145.134:1098     192.168.145.133:443       3376
0x81e562a8 192.168.145.134:1099     192.168.145.133:8888      4012
```

suspicious process identifiers in hand, we will now observe the victim host's process list with the "**pslist**" utiliy. Figure 2.3 shows us an abbreviated list of the running processes that were on the victim host, including each of their parent processes. Starting with process 3376, which appears to have spawned from the putty.exe application, we see that it corresponds to our outbound connection on port 443. This in itself is nothing more than just odd. This program has every right and ability to make network connections to a desired port and destination address. However, we see that its process, 3376, spawned another process directly below it, process 1136. It appears that our putty.exe application invoked a Windows command script. These are usually created with .js or .vbs extensions. We will soon address this anomaly for its own sake. Even more troubling, we see that process 1136, the command script, spawned process 4012 directly below it. Process 4012, as we recall, was associated with a suspicious network connection over port 8888. This likely indicates that we are dealing with an installed service designed to open a remote network connection. Another important note is that process 4012 is the ONLY svchost.exe process that was NOT spawned with services.exe as its parent process, making this all the more suspicious. With these leads in our examination, we are now almost certainly on the trail of some form of malicious activity and must investigate into suspicious registry key entries, the scripts they are associated with, and extract the originating executable, puttyx.exe. In this scenario, we will use the "**printkey**" utility to examine a popular malware registry key location. We see in Figure 2.4 that, as expected, a Windows registry

Figure 2.3: Now having knowledge of the offending processes, we search for their origin. We see below that a single executable, puttyx.exe, appears to be the root cause of two subsequent suspicious processes, a command script that invokes a backdoor network connection, and a the service that runs the network connection, though it was not spawned by services.exe.

Offset(V)	Name	PID	PPID	Thds	Hnds
0x825c8830	System	4	0	56	273
0x820ed020	smss.exe	552	4	3	21
0x820ad020	csrss.exe	616	552	11	417
0x823fbb10	winlogon.exe	640	552	18	555
0x82070890	services.exe	684	640	15	285
0x8231e3b8	lsass.exe	696	640	17	334
0x82105198	svchost.exe	876	684	18	202
0x82532020	svchost.exe	976	684	10	297
0x82527770	svchost.exe	1084	684	57	1133
0x82027020	svchost.exe	1200	684	4	72
0x82029da0	svchost.exe	1316	684	14	208
0x81f241f0	puttyx.exe	3376	1612	1	123
0x820d8ce0	cscript.exe	1136	3376	3	131
0x81ce11a0	svchost.exe	4012	1136	1	119

key has been discovered that points to a .vbs script, our cscript.exe on process 1136, located in a "Temp" folder. This is sufficient information to spur a larger investigation and begin automated remedial action. Finally, we used the "**procexedump**" utility to take a direct copy of the offending

Figure 2.4: We have discovered a registry key entry designed to re-hook the victim host upon reboot. We are also able to see the location that the .vbs script is stored in.

```
Legend: (S) = Stable   (V) = Volatile

----------------------------
Registry: \Device\HarddiskVolume1\WINDOWS\system32\config\software
Key name: Run (S)

Subkeys:

Values:
REG_SZ        VMware Tools    : (S) "C:\Program Files\VMware\VMware Tools\VMwareTray.exe"

REG_SZ        VMware User Process : (S) "C:\Program Files\VMware\VMware Tools\vmtoolsd.exe" -n vmusr

REG_SZ        bTSKObjWfQqQ    : (S) C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp\acxGcfWqt.vbs
```

executable, puttyx.exe, from memory for more in-depth analysis. One great benefit of taking this executable from memory is that it will be in its decoded/decrypted form, just as it was while running in memory. Further, if the encoding/decoding is self-contained in the executable, we will also have obtained the key/encoding scheme.

While this forensics example is relatively simple, it illustrates that known forensic methods can be emulated and that scripting with a powerful third-party tool could significantly enhance our ability to collect rich information for evidence preservation, attribution, to better protect our network, and motivates an Active Security architecture. Numerous other forensic tools allow for the same functionality and memory forensics are only one facet of digital forensics. Volatility itself has many more utilities that may have been used, but may not have been required for our specific implementation.

Chapter 3

Dynamic Network Modification with ClosedFlow

Software-Defined Networking substantially lowers the barrier for adding new networked services, such as flexible access control [26], web server load-balancing [35, 58], energy-efficient networking [36], adaptive network monitoring [39], and seamless user mobility and virtual-machine migration [30]. A software-defined network runs these services on a logically-centralized controller that uses a standard API (such as OpenFlow [46]) to install packet-handling rules in the underlying switches. In addition, SDN naturally supports network virtualization, where each virtual topology runs its own controller applications tailored to the needs of a particular "tenant" or class of traffic [55, 1, 7]. SDN can also enhance network scalability and reliability by leveraging advances in distributed systems to manage network state separately from any application-specific control logic [42]. Finally, and potentially most importantly, SDN can greatly enhance network security through the ability to dynamically adapt to changing threats [56, 34].

Despite the many advantages and possibilities of SDN, enterprises are understandably slow to adopt the new technology. For network administrators that want to move to a software-defined network today, the main option is effectively to 'fork lift' their existing infrastructure to SDN-capable devices (illustrated in the top half of Figure 3.1). Once they fork-lift replace their hardware, they can gradually transition over to SDN control with 'ships-in-the-night' behavior supported in most commercially available SDN switches, where traffic can be designated to be handled by either by the SDN-capable portion or by legacy protocols. In one attempt at an alternate transition, Panopticon effectively proposes sprinkling a few SDN switches (at the edge) and configuring the legacy

switches to act as tunnels between the SDN switches [44]. While this allows SDN functionality to be incorporated, it does require new hardware along with specialized configuration on the legacy devices to handle operating in support of SDN enabled switches.

We propose a different transition with ClosedFlow, illustrated in the bottom half of Figure 3.1, which allows SDN control over existing legacy hardware. That is, with no new hardware purchases an SDN controller, such as Floodlight [14] or Ryu [8], can run network controlling applications but, instead of targeting OpenFlow switches, legacy devices can be targeted (with no modifications to the applications, only to the code that interfaces with hardware).

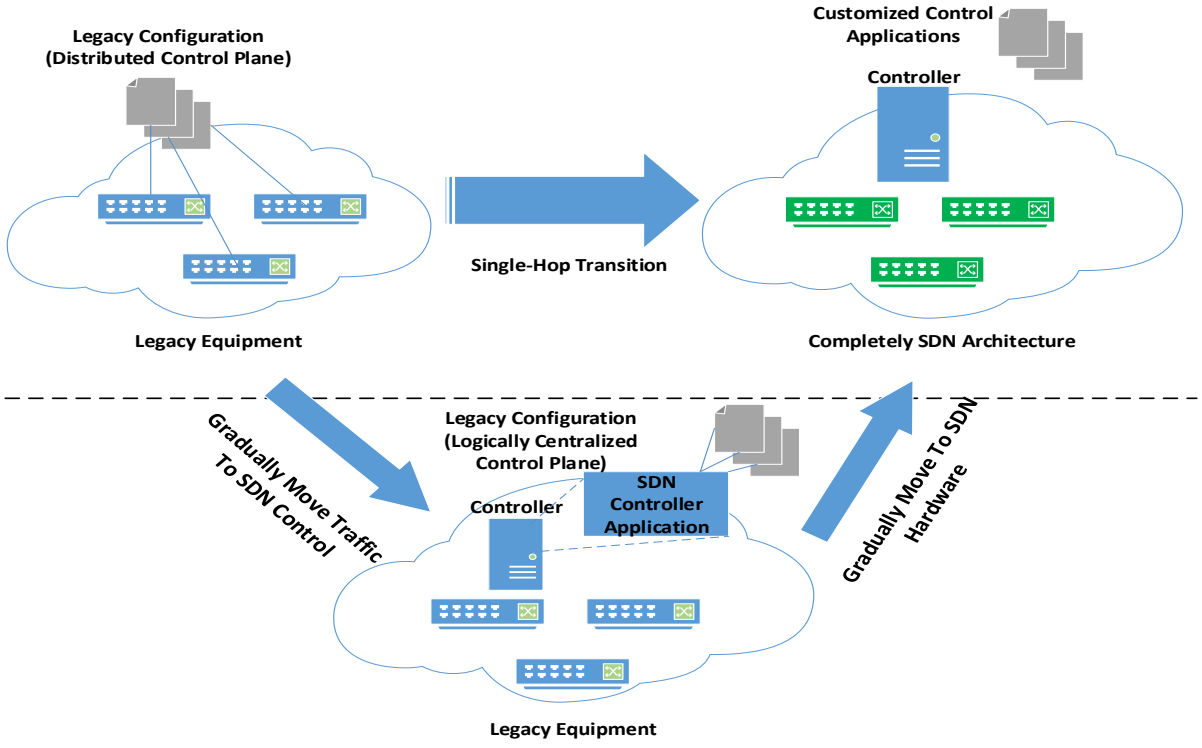
While many have been 'pushing configs' from a centralized server for years, the key question is how closely we can mimic the current embodiment of SDN (OpenFlow) with the proprietary configuration interfaces of legacy devices. We make the following contributions.

- We explore and answer this key question by demonstrating the four main capabilities in OpenFlow: establishing a control channel, automatically discovering the topology for a network-wide view, modifying a flow table with entries that specify matching packet headers and actions to forward or drop packets, and handling the 'send-to-controller' actions. We demonstrate these four capabilities, though handling the 'send-to-controller' action requires a compromise. Our prototype **static flow pusher** application is also discussed.

- We evaluate the system with our prototype targeting 10 year old Cisco switches. We show that while the table sizes of legacy switches is not as large as modern SDN switches, the legacy switches have flow table optimization technology which allows for many flow table entries to be set before processing needs to be relegated to lookups in slower memory such as DRAM.

- We illustrate that if we don't limit ourselves to OpenFlow, we can enable much more capabilities by tapping into functionality that has been present in commercial switches for over a decade.

Figure 3.1: Proposed below is an alternative transition of a legacy network with a control plane that is distributed amongst many devices across the network to one with logically central, SDN control.



3.1 Realizing ClosedFlow

The key question is whether we can provide an OpenFlow interface (from the SDN controller's perspective) to control a network of legacy switches. That is, we are not just proposing a centralized interface to legacy switches, with vendor specific configurations. We are proposing that the control associated with OpenFlow (and capitalized on in the many research works illustrating the benefits), can be achieved with legacy switches and with hardware performance.

To explore this, in each of the subsections we will illustrate the four main characteristics of an OpenFlow network: (i) A communication channel between a central controller and each switch, (ii) topology discovery, (iii) matching packets up to layer 4 and applying standard actions (drop, forward), and (iv) handling the special action 'send-to-controller' with packet-in messages. Here, we focus on Cisco configuration for switches with a with a minimum IOS of 12.2(44)SE (which we have running on our over 10 year old Cisco 3550 switches), but we note that similar capabilities are present in all major vendor's switches.

3.1.1 Controller-switch Control Channel

A central requirement in SDN is that a centralized controller is able to communicate with each switch in the network and not need to be physically (directly) connected to each switch. Ethane overcame this through the use of spanning-tree protocol, commonly used to support plug-and-play Ethernet networks.

One challenge we face is that we are asking each switch in our topology to operate over layer-3 interfaces (in order to support matching at layer's 3 and 4). Because of this, we do not have layer-2 protocols, namely spanning-tree protocol, at our disposal to automatically discover and calculate paths between each switch and the controller (for control traffic).

To facilitate a communication channel and topology state awareness between switches and our controller, we chose to run a minimal instance of the Open-Shortest-Path-First (OSPF) routing protocol. To be clear, this instance is segregated from data flow traffic, which is forwarded based

on route-map configurations and access-list match conditions. Our OSPF instance includes advertisements for the Loopback management interfaces of each switch, a point-to-point connection between switches, and a VLAN dedicated for communication with the controller(s). With this, we enabled an in-band overlay control channel and remote access (SSH or telnet) to each switch for control traffic, such as pushing new flow rules.

New switch installation does require some minimum configuration. For our architecture, the below basic configurations are sufficient for the installation of a new switch.

- **Set IP address for interface Loopback 0**

```
SW1(config-if)# ip address 1.2.3.4 255.255.255.255
```

- **Configure "routed" interfaces for switch-to-switch links**

```
SW1(config)# interface f0/24
SW1(config-if)# no switchport
SW1(config-if)# ip address 2.0.0.1 255.255.255.252
```

- **Configure OSPF Instance and Set Router-ID to Loopback 0 IP**

```
SW1(config)# router ospf 1
SW1(config-router)# router-id 1.2.3.4
SW1(config-router)# log-adjacency changes
```

- **Advertise Loopback and point-to-point networks in OSPF**

```
SW1(config-router)# network 1.2.3.4 0.0.0.1 area 0
SW1(config-router)# network 2.0.0.0 0.0.0.3 area 0
```

- **Set up remote access (SSH or Telnet)**

```
SW1(config)# line vty 0 4
SW1(config-line)# password Remote
SW1(config-line)# login
```

- **Set Enable Mode Password**

```
SW1(config)# enable secret Cisco
```

This effectively establishes the line of communication with our controller, and enables remote access facilities to allow the controller to push configurations to each device.

3.1.2 Topology Discovery

A second requirement for an SDN network is for the controller to have a network-wide view, including a view of the entire topology. With Ethane, this was achieved by each switch periodically sending link-state information to the controller.

In our architecture, we could also follow Ethane's approach. The first is to use remote logging to the controller from each switch, to simply log OSPF adjacency changes. This would allow the controller to store topology state and maintain its control channel in the event of a link failure.

Alternatively, we can capitalize on the fact that in OSPF, each node in the network has complete visibility over the entire topology. This would require the controller to run the complete OSPF routing protocol and expose the current topology to the SDN controller. We believe the first approach is simpler and lighter-weight, though we have not implemented the second approach to say with certainty.

Our stored topology information will serve as an important reference point for the controller to know what route-map to apply to a specific interface. We also store ACL and route-map configuration information for each switch to keep track of existing match rules and route-map sequence numbers.

3.1.3 Packet Matching and Applying Actions

The third requirement is that the SDN controller have the fine-grain ability to control how individual flows are handled, as opposed to tuning knobs on a variety of standard routing protocols. In an OpenFlow network, flow rules along with the forwarding behavior can be defined fairly simply.

In targeting legacy switches, we can achieve a similar level of control with a combination of access-control lists (ACLs), route-maps, and interface configuration.

- **Access control list** – ACLs specify the match conditions to classify the packet, and whether to permit or deny the traffic. If the OpenFlow action is to drop, that is set here. If the action is to forward out a specific port, the ACL will be configured to permit and a route map will handle the forwarding behavior.

- **Route Map** – Route-maps are applied at an inbound interface, and specifies several pairs of an ACL to match on and the forwarding behavior.

- **Interface** – specifies which route map is relevant to it. Each physical interface, such as FastEthernet 0/48, can have a different Route Map, or if groups of physical interfaces are to have their traffic handled in the same manner (if the OpenFlow rule does not specify an input port as part of the match condition), then we can optimize by assigning the interfaces all to the same VLAN, so that they are all aggregated as one interface.

To illustrate, shown below is some familiar OpenFlow syntax that is used to describe match criteria and apply a forwarding behavior. We expect to enter a flow with something like the below statement.

```
match: src_ip="1.2.3.4", dest_ip="2.3.4.5", action:OUT_PORT_2
```

To push a flow rule and its forwarding behavior to a switch we want to create a route-map, apply an access-control list to it, define its forwarding behavior, install an access-control entry for

the access-list to match on, and finally, apply the route-map to the interfaces we require as given by our stored topology state.

The ClosedFlow controller application creates the new route-map and defines forwarding behavior as the next-hop IP address of our adjacent switch (per the topology discovery, we know OUT_PORT_2 has a next hop address of 2.0.0.1, applies an access-list to it (ACL 101), creates the access-control entry based on the operator supplied match condition (involving the source and destination addresses), and applies the route-map to interface VLAN 1 (in this case, no match condition specifies the input port, so we default to using the default VLAN interface). Configurations applied shown below.

```
Switch1#show route-map
route-map SW1_OUTBOUND, permit, sequence 10
  Match clauses:
    ip address (access-lists): 101
  Set clauses:
    ip next-hop 2.0.0.1

Switch1#show access-lists
Extended IP access-list 101
    10 permit ip host 1.2.3.4 host 2.3.4.5

Switch1#show run interface vlan 1
interface Vlan1
 ip address 1.2.3.1 255.255.255.0
 ip policy route-map SW1_OUTBOUND
end
```

3.1.4 Handling Packet-In Events

The fourth capability associated with OpenFlow networks is the special action 'send to controller', which can be used, for example, to enable a reactive network where the first packet of every flow is sent to the controller.

In an OpenFlow network, when a packet arrives at an interface, we have a couple potential outcomes: the packet matches a flow table entry and some forwarding, dropping, or modify action is applied, or when the packet does not meet these criteria (a "table miss"), it is sent to the controller allowing decision logic to take place. We propose two potential options to handle "table misses", or more generally 'send-to-controller' actions (which do not have to be table misses) – neither option mimics OpenFlow exactly, but has aspects of OpenFlow's two ways to handle send-to-controller.

- **Remote Logging on Explicit Deny:** Our first option uses the remote logging feature to send a message to the controller when a packet does not match our access control criteria specified in our route-map. To achieve this, we must place an "explicit deny" access control entry (ACE) at the end of our access-control list. Additionally, we must us the "log-input" keyword to indicate that a syslog entry should be made if the explicit deny is matched on. To ensure that our remote logging is restricted to this specific message, we create a logging discriminator that uses regular expression matching and we suppress excessive logging with simple threshold limits until a flow rule is installed. An example of these configuration steps is shown below, and the down side is that while the header of the packet is sent to the controller, the packet itself is dropped (instead of being buffered at the switch, as in OpenFlow).

```
SW1(config)# access-list 101 permit icmp host 2.2.2.2 host 3.3.3.3
SW1(config)# access-list 101 permit icmp host 3.3.3.3 host 2.2.2.2
SW1(config)# access-list 101 deny udp any gt 0 any gt 0 log-input
SW1(config)# access-list 101 deny tcp any gt 0 any gt 0 log-input
SW1(config)# access-list 101 deny ip any any log-input
SW1(config)# logging discriminator tblMiss msg-body ‘‘101 denied’’
SW1(config)# logging host <<controller-IP>> transport udp port 22345
    discriminator tblMiss
```

- **Send Entire Packet To Controller:** Rather than implementing remote logging to notify the controller that no match criteria were met, we also have the ability to direct the flow to the controller and allow it to decide whether the traffic will be forwarded over a particular interface or be dropped by the switches involved. To accomplish this, rather than logging an access control entry, we want to apply a “forward-to-controller” behavior by essentially defining this as our default forwarding behavior. Once the controller receives the first packet, a decision is made to install a flow entry to either forward the traffic down a preferred path, or drop the traffic.

```
SW1(config)# access-list 103 deny ip any any
SW1(config)# route-map SW1_OUT 15
SW1(config-route-map)# match ip address 103
SW1(config-route-map)# set ip next-hop <<controller-IP>>
SW1(config)# int vlan 1
SW1(config-if)# ip policy route-map SW1_OUT
```

Neither of these options, directly matches the OpenFlow packet-in message, specifically the option where the packet is buffered on the switch and only the header is sent to the controller. In one option, we can match the behavior that only the header is sent to the controller, but the packet is dropped (relying on retransmission). In the other option, we can match the behavior of

OpenFlow which sends the entire packet to the controller, which has overhead, but only for highly reactive networks (more proactive SDN control would not suffer from this overhead). We are still exploring mechanisms to realize the buffering capability to increase reliability.

3.2 Prototype

Our end-goal is for this to be able to be integrated into an SDN controller, and allow SDN applications to run, unmodified, controlling legacy switches instead of OpenFlow switches. Our initial prototype is not quite there yet. We instead implemented two independent programs to illustrate the key parts:

- A constantly-running topology discovery application which uses the info received from the remote logs to display the current adjacencies, and
- A simple python program equivalent to static flow pusher which allows flow modifications to be specified.

With these two capabilities, we intend to integrate a Cisco configuration backend with an SDN controller which provides an API which can closely match the OpenFlow control (with the OFPTableMod method in Ryu, or with the flow entry files in **yanc** [48]). Since **yanc** completely separates out the the hardware interfacing control, it provides a simple path to allowing us to add support for the legacy devices (by creating a new driver) with no modifications to the yanc code or applications – and we can, without modifying the application, we can gradually replace legacy switches with OpenFlow enabled switches.

3.3 Experiment and Evaluation

3.3.1 Environment Setup

Throughout our experiments and implementation, we utilized Cisco 3550 Multi-Layer Switches with a minimum IOS of 12.2(44)SE. Our research and observations have shown that functionality

and control only increase in granularity with each new evolution. In particular, we later examine Cisco's Embedded Event Manager and Tool Command Line scripting features, first introduced to the Catalyst family of switches, with the Cisco 3560 MLS using IOS version 12.2-(55)SE. These features enable vast potential for customized functionality to support communication with a central control plane.

- **Configure SDM Template:** To optimize our environment for policy-based routing and TCAM ACL entries, we must first reformat our TCAM table using the Switch Database Manager. Template options for formatting the TCAM tables include access, default (balance of all functionality), routing, and vlan. The first of these options, access, allows us to maximize our resources for ACL functionality. We choose this template because ACL entries on layer 3 and 4 fields will act as the majority of our configuration entries when installing flow rules. To enable policy-based routing it is important that the 'extended-match' keyword is used with the SDM template enabling 144-bit layer-3 TCAM. After these commands are used, a reboot is required.

- **Enable IP Routing and Cisco Express Forwarding**: Because we wish to match on layer 3 and 4 packet fields and define interface forwarding behavior with policy-based routing, we will enable IP routing and Cisco Express Forwarding (CEF). We exploit the benefits that CEF offers through its direct use of the Forwarding Information Base and adjacency tables to perform fast IP switching coupled with PBR Route-Maps that specify forwarding behaviors and match criteria stored in the TCAM.

3.3.2 Results

To evaluate the effects of our technique on the hardware and the feasibility of operation in a production network, we chose to measure the direct correlation between installed flow rules and their storage in the TCAM. We chose 3 flow rule datasets: a realistic enterprise sampling which used realistic IP address ranges, port ranges, and matching on both layer 3 and 4 fields.

The other two sample datasets used a worst-case, completely random source/destination IP and source/destination port combination. With these datasets, we essentially want to see how quickly we can cause TCAM ACL installation and merge failures if an administrator doesn't adhere to basic, responsible configuration best practices. In each worst case event, rules are applied in software, requiring CPU processing for match criteria.

We see that our realistic network flow dataset scales quite well. The Cisco Catalyst 3550 multi-layer switches used for our experiments permitted up to 4432 PBR TCAM entries. Based on the graph in Figure 3.2, we may extrapolate that our flow rule limit would be 50,000 for this particular piece of hardware. In our final two randomized datasets, we see that the switch begins to experience ACL merge and installation failures at 500, for layer 3, and 250 layer 4, access-list rules respectively. These results are consistent with Cisco's recommendations to avoid TCAM resource exhaustion.

For our next evaluation, we chose to determine the impact of the previous cases where we experienced TCAM merge and installation failures, that is, the newly added flow rules would be processed in software until space became available. We performed tests using Iperf to witness forwarding rate performance at 100 percent interface utilization. Figure 3.3 depicts the results, by average bitrate, for flow rules installed in software, as IOS configuration that are CPU processed, and in hardware, installed as TCAM policy-based routing entries. Two flows are depicted, one installed in hardware from time zero, and another that is installed in software (because the TCAM was too full), and at 10 seconds in, can be moved to hardware as space became available (we deleted other rules). Upon transition into TCAM, we see an immediate jump to full hardware speed.

Figure 3.2: Match conditions are defined as Cisco IOS access-control list entries. When an access-list is applied to an active route-map on an interface, these match conditions are installed as policy-based route entries directly in the TCAM. This graph illustrates the correlation between configured IOS access-control entries, and the amount of entry space in the TCAM that they occupy.

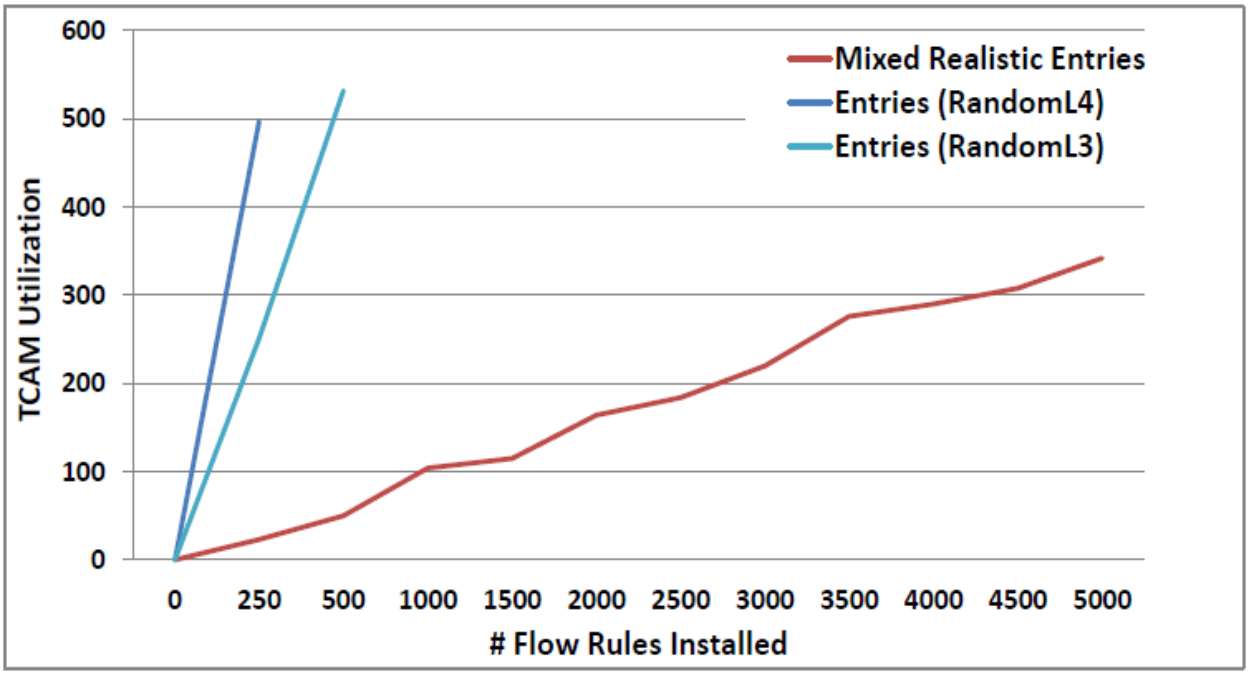
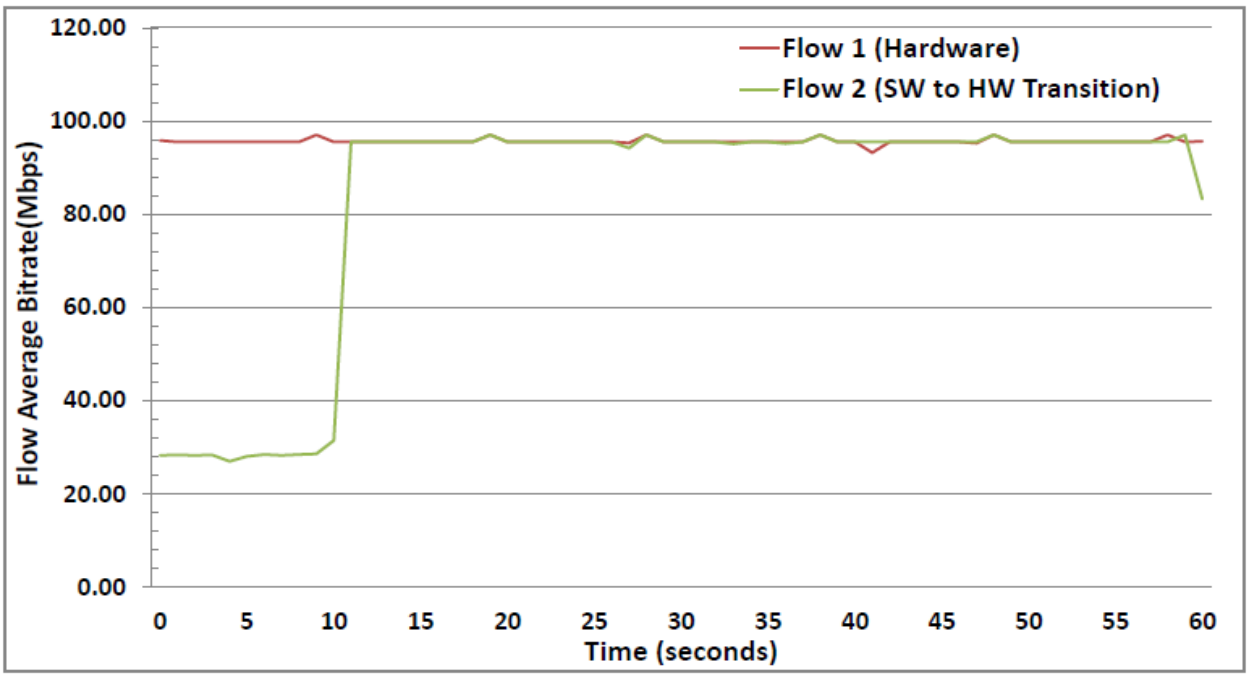


Figure 3.3: This graph depicts the performance of different flows based on whether they are installed in software, where they are CPU processed, or in hardware, where TCAM lookups are performed and full hardware speed is achieved. Flow two illustrates the instantaneous transition from software to hardware when TCAM space becomes available.



Chapter 4

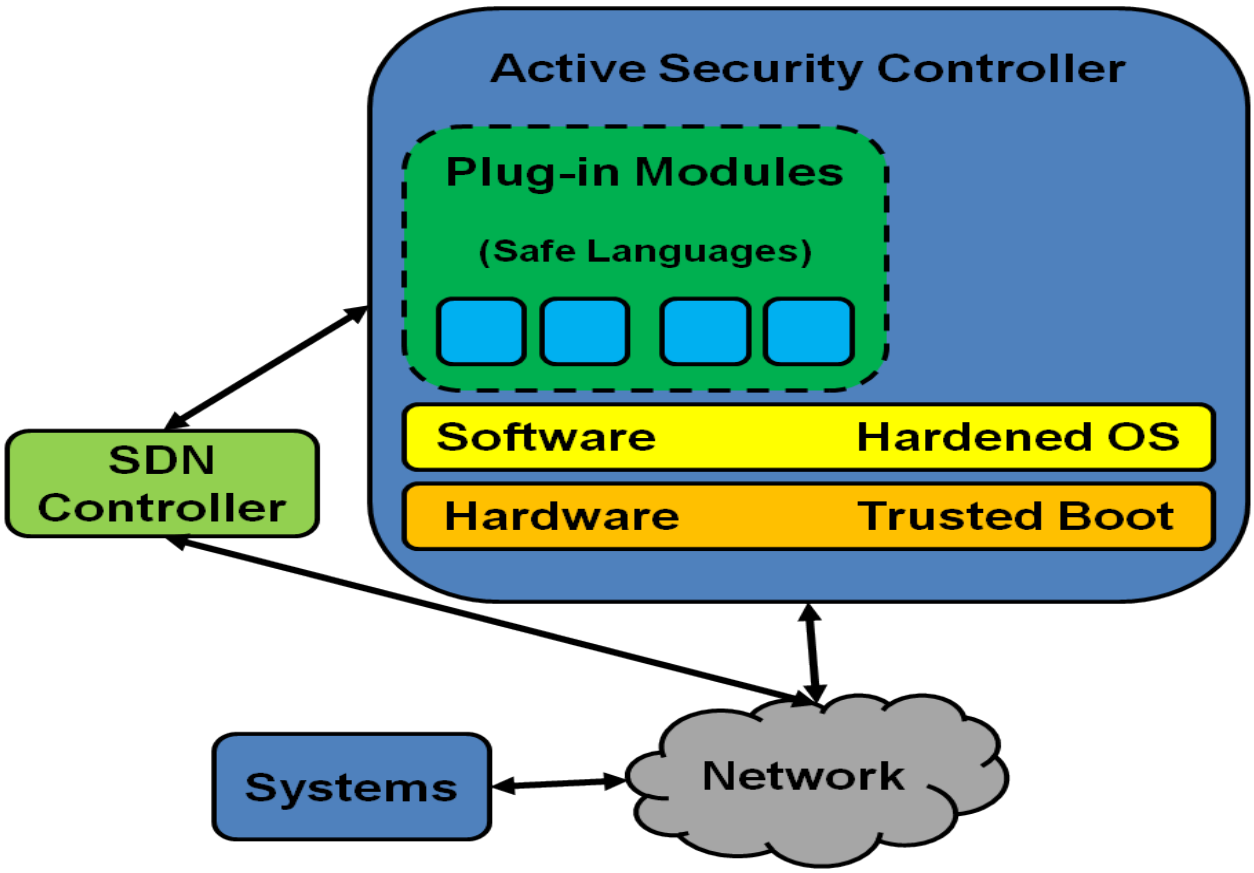
Future Work and Research

There are numerous areas of future research that we have identified and taken into consideration while working on Active Security, forensics, and ClosedFlow.

4.1 Securing the Controller

Active security introduces a software program within a feedback loop that monitors a networked infrastructure and takes actions on the infrastructure to better monitor and protect it. This introduces a new source of vulnerability that must be dealt with. In particular, (i) the privileged controller at the center of the control loop, and (ii) the interfaces to the various actions and sensors in the infrastructure. For the controller, the upshot is that there is only one (or a few) of these that need to be protected, and we can narrow our focus on securing that one system. We believe that much of the concern can be addressed through a combination of existing technology such as trusted boot [21], operating systems [40, 43, 29, 62], languages [24, 18, 49], and SDN controllers [25, 53]. For the interfaces, we will make use of network-based enforcement and monitoring. That is, we plan to investigate an approach that will restrict access to actions/sensors to those coming from the controller (we can verify the entire path with the underlying programmable network) and we will monitor/log all message exchanges.

Figure 4.1: A natural uneasiness comes from the fact that we have placed a high level of trust and control, in seemingly few places. We believe that we can leverage existing technologies to address this issue.



4.2 Deployment (BYOD)

In the running example, we presented a situation where an enterprise has full control over the end hosts, e.g., being able to deny the device access to the network without a specific software configuration. Today, employees want to bring their own devices to the work place. This bring your own device (BYOD) challenge is an open issue for enterprises. Without the ability to run software on end systems (such as mobile phones), we clearly cannot execute software to, for example, obtain an snapshot of memory. We can, however, still actively manage the rest of the security response. For example, SDN has been applied to be able to isolate personal devices on a campus network [13]. Further, virtualization is being introduced as a means to allow employees to bring their own device, but employers to control the software environment [19, 10]. It might be possible to leverage that to be able to execute software on the mobile device without impacting the users personal VM.

4.3 Stealthy and Efficient Forensics

Importantly, our forensic evidence gathering needs to be performed in a stealth manner, so that the attacker cannot detect it and in response, clean up. BodySnatcher [12], was one technique which injects code to gain full control over the OS, in order to get a memory dump reliably, not necessarily stealthily. There have been a few research projects which indicate we will be able to perform volatile memory gathering in a stealth manner. CoPilot [52] used special functionality on a PCI care to be able to read memory contents without any software involvement at all. Likewise, HyperSentry [22] used the System Management Mode (SMM) and the Intelligent Platform Management Interface (IPMI) to be able to trigger the execution of code to measure with the trusted platform module (TPM) without being detected by the malicious code. Of course, in each case we need to ensure the security of these open interfaces, as evident with IPMI related vulnerabilities. Here, network based protection is especially important as were unlikely to be able to fix all embedded systems. We believe memory to be one of the more challenging sources of evidence to gather in a stealth manner, and believe other mechanisms will be possible over the

course of the research. Moreover, the forensic gathering needs to be efficient. Naively, for example, grabbing an 8GB dump of volatile memory when our active security controller thinks an attack may be happening, can consume quite a bit of network bandwidth and take a long time (especially if we are attempting to go undetected by the attacker). Instead, we need a more focused forensic evidence gathering procedure which is an iterative exploration of the available data. This will require some analysis, and operating system specific procedures.

4.4 Expanded Sensor Diversity

A well-rounded approach to network security must consider many different vantage points to gain a complete picture of what is actually happening. At times, this information can become so complex and often too sensitive that it feels like looking for a needle in a haystack. Luckily the community of security professionals have improved at pruning off the excess and fine tuning their sensors. Our active security controller helps take this a step further by using programmed responses to act on an attack or anomalous traffic in real-time while attempting to maintain resource availability. The modular nature of this concept and mechanism also facilitates the incorporation of virtually any tool or technique. This means that, even if an organization is confident in the security tools and products they are using, an active security controller will enhance their effects, not replace them.

4.5 OpenFlow Extensions

In some regards we are able to go beyond what OpenFlow provides if we allow capabilities of the legacy switches to be exposed (if we don't limit ourselves to OpenFlow only). For example, AvantGuard [57], DevoFlow [27], and Software-defined counters [47] each noted some limitations with OpenFlow with regard to security or monitoring applications and proposed new switch additions. Existing legacy switches already have lots of other capabilities, such as the on-switch monitoring with triggered events along the lines of what AvantGuard and Software-defined counters proposed, namely through the use of Cisco's Embedded Event Manager [3]. With the send

to controller action, we were not able to exactly match what OpenFlow can provide. But with legacy switches we can match the intent of the proposed extensions (though, likely not the exact realization proposed).

4.6 Other Switches (Vendors, Models)

Despite the fact that we chose to implement these techniques in Cisco 3550 and 3560 series multi-layer switches, we note the presence of identical functionality among other major vendors' operating systems – we examined HP and Juniper switches, specifically, and believe we would find it to be the case with other vendors as well.

Further, it is important to note that the functionality we describe can be realized in newer equipment models (from Cisco). Each generation beyond the equipment we used carries richer functionality and more powerful features than the previous. In slightly later model Cisco switches, we have the facility for logging Cisco Discovery Protocol adjacency changes or the use of the Link Layer Discovery Protocol at layer-2 for topology discovery which allows us to avoid using OSPF for control channel connectivity and communication at time-zero. In many older Cisco branch routers and slightly newer multi-layer switches, we have added packet classification granularity with the Network Based Application Recognition(NBAR) feature. This allows us to use deep packet inspection to classify traffic and make decisions up to layer-7. Cisco maintains a vast database of application signatures and NBAR allows for creating custom signatures.

Chapter 5

Conclusion

This thesis presented a high-level view of a new Active Security system inspired by an Observe, Orient, Decide, Act, feedback loop that leverages the programmatic control of Software-Defined Networking. We discuss many of the benefits that this architecture can offer and motivate its use through the case of in-attack forensic data collection to attribute attacks, preserve critical evidence, and ultimately use as input back into our feedback loop to better protect the network. With SDN as a requirement to fully realize such a system, and enterprise networks being an ideal consumer, we recognized the relatively slow adoption of SDN as an obstacle. ClosedFlow shows that there is great potential for introducing SDN control in legacy networks with proprietary equipment today and permitting a gradual and budget friendly hardware transition after the fact. The simple fact is that security must break this cat-and-mouse game with the adversary and attempt to disrupt their exploit development and attack life-cycle with an architecture that forces them to be highly reactive and gives the defender the advantage.

Bibliography

[1] About nicira. http://www.nicira.com/about/.

[2] Arbor Networks Peakflow SP Threat Management System. http://www.arbornetworks.com/products/peakflow/tms.

[3] Cisco IOS Embedded Event Manager (EEM). http://www.cisco.com/c/en/us/products/ios-nx-os-software/ios-embedded-event-manager-eem/.

[4] Immunity debugger. http://www.immunityinc.com/products-immdbg.shtml.

[5] Nikto2. https://www.cirt.net/Nikto2.

[6] OWASP Zed Attack Proxy Project. https://www.owasp.org/index.php/OWASP$_Zed_Attack_Proxy_Project$.

[7] Perspectives: Networking needs a VMware. http://www.bigswitch.com/wp/about-us/.

[8] Ryu. http://osrg.github.io/ryu/.

[9] Solarwinds. http://www.solarwinds.com/log-event-manager/active-response-library.aspx.

[10] VMware Horizon Mobile. http://www.vmware.com/products/desktop$_virtualization/mobile/overview.html$.

[11] The volatility framework: Volatile memory artifact extraction utility framework. https://www.volatilesystems.com/default/volatility.

[12] Bodysnatcher: Towards reliable volatile memory acquisition by software., 2007.

[13] Ballarat grammar secures byod with hp sentinel sdn. http://h20195.www2.hp.com/v2/GetPDF.aspx/4AA4-7496ENW.pdf, Aug. 2013.

[14] Floodlight. http://floodlight.openflowhub.org, 2013.

[15] Linux memory extrator. https://code.google.com/p/lime-forensics/, 2013.

[16] Snort. http://www.snort.org/, 2013.

[17] Greg Abelar and Dale Tesch. Role of CS-MARS in Your Network. http://www.ciscopress.com/articles/article.asp?p=664149, 2006.

[18] D. Scott Alexander and Jonathan M. Smith. The Architecture of ALIEN. In Proc. International Working Conference on Active Networks (IWAN), 1999.

[19] Jeremy Andrus, Christoffer Dall, Alexander Van't Hof, Oren Laadan, and Jason Nieh. Cells: a virtual mobile smartphone architecture. In Proc. Symposium on Operating Systems Principles (SOSP), 2011.

[20] Dirk Grunwald Andy Sayler, Eric Keller. Jobber: Automating inter-tenant trust in the cloud. In Proc. Workshop on Hot Topics in Cloud Computing (HotCloud), 2013.

[21] W. A. Arbaugh, D. J. Farber, and J. M. Smith. A secure and reliable bootstrap architecture. In Proc. IEEE Symposium on Security and Privacy, 1997.

[22] Ahmed M. Azab, Peng Ning, Zhi Wang, Xuxian Jiang, Xiaolan Zhang, and Nathan C. Skalsky. Hypersentry: enabling stealthy in-context measurement of hypervisor integrity. In Proc. ACM conference on Computer and communications security (CCS), 2010.

[23] Yair Bartal. Firmato: A novel firewall management toolkit. Proceedings of the 1999 IEEE Symposium on Security and Privacy, 22(4):381–420, 2004.

[24] B. N. Bershad, S. Savage, P. Pardyak, E. G. Sirer, M. E. Fiuczynski, D. Becker, C. Chambers, and S. Eggers. Extensibility safety and performance in the spin operating system. In Proc. symposium on Operating systems principles (SOSP), 1995.

[25] Marco Canini, Daniele Venzano, Peter Peresini, Dejan Kostic, and Jennifer Rexford. A NICE way to test OpenFlow applications. In Proc. Network System Design and Implementation (NSDI), Apr. 2012.

[26] Martin Casado, Michael J. Freedman, Justin Pettit, Jianying Luo, Nick McKeown, and Scott Shenker. Ethane: taking control of the enterprise. In Proc. SIGCOMM, 2007.

[27] Andrew R. Curtis, Jeffrey C. Mogul, Jean Tourrilhes, Praveen Yalagandula, Puneet Sharma, and Sujata Banerjee. DevoFlow: Scaling Flow Management for High-performance Networks. In Proc. SIGCOMM, 2011.

[28] Brendan Dolan-Gavitt. The VAD tree: A process-eye view of physical memory. Digital Investigation, 4:62–64, 2007.

[29] Petros Efstathopoulos, Maxwell Krohn, Steve VanDeBogart, Cliff Frey, David Ziegler, Eddie Kohler, David Mazières, Frans Kaashoek, and Robert Morris. Labels and event processes in the asbestos operating system. In Proc. symposium on Operating systems principles (SOSP), 2005.

[30] David Erickson et al. A demonstration of virtual machine mobility in an OpenFlow network, August 2008. Demo at ACM SIGCOMM.

[31] Simson Garfinkel. The criminal cloud. http://www.technologyreview.com/news/425770/the-criminal-cloud/, Oct 2011.

[32] Albert Greenberg, Gisli Hjalmtysson, David A. Maltz, Andy Myers, Jennifer Rexford, Geoffrey Xie, Hong Yan, Jibin Zhan, and Hui Zhang. A clean slate 4d approach to network control and management. SIGCOMM Comput. Commun. Rev. (CCR), 35(5):41–54, Oct. 2005.

[33] Andreas Haeberlen, Paarijaat Aditya, Rodrigo Rodrigues, and Peter Druschel. Accountable virtual machines. In Proc. USENIX conference on Operating systems design and implementation (OSDI), 2010.

[34] Ryan Hand, Michael Ton, and Eric Keller. Active security. In Proc Workshop on Hot Topics in Networks (HotNets), 2013.

[35] Nikhil Handigol, Srinivasan Seetharaman, Mario Flajslik, Nick McKeown, and Ramesh Johari. Plug-n-Serve: Load-balancing web traffic using OpenFlow, August 2009. Demo at ACM SIGCOMM.

[36] Brandon Heller, Srini Seetharaman, Priya Mahadevan, Yiannis Yiakoumis, Puneet Sharma, Sujata Banerjee, and Nick McKeown. ElasticTree: Saving energy in data center networks. April 2010.

[37] Jafar Haadi Jafarian, Ehab Al-Shaer, and Qi Duan. Openflow random host mutation: transparent moving target defense using software defined networking. In Proc. Workshop on Hot topics in software defined networks (HotSDN), 2012.

[38] Sushil Jajodia, Anup K. Ghosh, Vipin Swarup, Cliff Wang, and Xiaoyang Sean Wang, editors. Moving Target Defense - Creating Asymmetric Uncertainty for Cyber Threats, volume 54 of Advances in Information Security. Springer, 2011.

[39] Lavanya Jose, Minlan Yu, and Jennifer Rexford. Online measurement of large traffic aggregates on commodity switches. In Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services, March 2011.

[40] Gerwin Klein, Kevin Elphinstone, Gernot Heiser, June Andronick, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, et al. seL4: Formal verification of an OS kernel. In Proc. symposium on Operating systems principles (SOSP), 2009.

[41] Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti, and M. Frans Kaashoek. The Click modular router. ACM Transactions on Computer Systems, 18(3):263–297, August 2000.

[42] Teemu Koponen, Martin Casado, Natasha Gude, Jeremy Stribling, Leon Poutievski, Min Zhu, Rajiv Ramanathan, Yuichiro Iwata, Hiroaki Inoue, Takayuki Hama, and Scott Shenker. Onix: A distributed control platform for large-scale production networks. October 2010.

[43] Maxwell Krohn, Alexander Yip, Micah Brodsky, Natan Cliffer, M. Frans Kaashoek, Eddie Kohler, and Robert Morris. Information flow control for standard os abstractions. In Proc. symposium on Operating systems principles (SOSP), 2007.

[44] Dan Levin, Marco Canini, Stefan Schmid, and Anja Feldmann. Panopticon: Reaping the benefits of partial sdn deployment in enterprise networks. Technical report, Technische Universität Berlin / Deutsche Telekom Laboratories, 2013.

[45] Chia-Chi Lin, Matthew Caesar, and Jacobus Van der Merwe. Towards Interactive Debugging for ISP Networks. In ACM Workshop on Hot Topics in Networks (HotNets), Oct. 2009.

[46] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. OpenFlow: Enabling innovation in campus networks. SIGCOMM Comput. Commun. Rev. (CCR), 38(2), 2008.

[47] Jeffrey C. Mogul and Paul Congdon. Hey, You Darned Counters!: Get off My ASIC! In Proc. Workshop on Hot Topics in Software Defined Networks (HotSDN), 2012.

[48] Matthew Monaco, Oliver Michel, and Eric Keller. Applying Operating System Principles to SDN Controller Design. In Proc. Workshop on Hot Topics in Networks (HotNets), 2013.

[49] Santosh Nagarakatte, Jianzhou Zhao, Milo M.K. Martin, and Steve Zdancewic. SoftBound: highly compatible and complete spatial memory safety for c. In Proc. conference on Programming language design and implementation (PLDI), 2009.

[50] Santosh Nagarakatte, Jianzhou Zhao, Milo M.K. Martin, and Steve Zdancewic. CETS: compiler enforced temporal safety for C. In Proc. international symposium on Memory management (ISMM), 2010.

[51] RICHARD Perez-Pena. Universities face a rising barrage of cyberattacks. http://www.nytimes.com/2013/07/17/education/barrage-of-cyberattacks -challenges-campus-culture.html, Jul 2013.

[52] Nick L. Petroni, Jr., Timothy Fraser, Jesus Molina, and William A. Arbaugh. Copilot - a coprocessor-based kernel runtime integrity monitor. In Proc. USENIX Security Symposium, 2004.

[53] Philip Porras, Seungwon Shin, Vinod Yegneswaran, Martin Fong, Mabry Tyson, and Guofei Gu. A security enforcement kernel for OpenFlow networks. In Proc. workshop on Hot topics in software defined networks (HotSDN), 2012.

[54] Michael Riley and Ben Elgin. Chinas Cyberspies Outwit Model for Bonds Q. http://www.bloomberg.com/news/2013-05-01/china-cyberspies-outwit-u-s-stealing -military-secrets.html, May 2013.

[55] Rob Sherwood, Glen Gibb, Kok-Kiong Yap, Guido Appenzeller, Martin Casado, Nick McKeown, and Guru Parulkar. Can the production network be the testbed? October 2010.

[56] Seungwon Shin, Phil Porras, Vinod Yegneswaran, Martin Fong, Guofei Gu, and Mabry Tyson. Fresco: Modular composable security services for software-defined networks. In Proc. Network and Distributed System Security Symposium (NDSS), February 2013.

[57] Seungwon Shin, Vinod Yegneswaran, Phillip Porras, and Guofei Gu. Avant-guard: Scalable and vigilant switch flow management in software-defined networks.

[58] Richard Wang, Dana Butnariu, and Jennifer Rexford. OpenFlow-based server load balancing gone wild. In Hot-ICE, March 2011.

[59] Xitao Wen, Yan Chen, Chengchen Hu, Chao Shi, and Yi Wang. Towards a secure controller platform for openflow applications (extended abstract). In Proc. workshop on Hot topics in software defined networking (HotSDN), 2013.

[60] A Wool. A quantitative study of firewall configuration errors. Computer, 37:62–67, 2004.

[61] A. Wundsam, D. Levin, S. Seetharaman, and A. Feldmann. Ofrewind: enabling record and replay troubleshooting for networks. In USENIX Annual Technical Conference, 2011.

[62] Nickolai Zeldovich, Silas Boyd-Wickizer, Eddie Kohler, and David Mazières. Making information flow explicit in histar. In Proc. symposium on Operating systems design and implementation (OSDI), 2006.