Active Security

Ryan Hand, Michael Ton, Eric Keller,
University of Colorado, Boulder
{ryan.hand, michael.ton, eric.keller}@colorado.edu

ABSTRACT

In this paper we introduce *active security*, a new methodology which introduces programmatic control within a novel feedback loop into the defense infrastructure. Active security implements a unified programming environment which provides interfaces to (i) protect the infrastructure under common attack scenarios (*e.g.,* configure a firewall), (ii) sense the current state of the infrastructure through a wide variety of information, (iii) adjust the configuration of the infrastructure at run time based on sensed information, (iv) collect forensic evidence on-demand, at run-time for attribution, and (v) counter the attack through more advanced mechanisms such as migrating malicious code to a quarantined system. We built an initial prototype that extends the FloodLight software-defined networking controller to automatically interface with the Snort intrusion detection system to detect anomalies, the Linux Memory Extractor to collect forensic evidence at run-time, and the Volatility parsing tool to extract an executable from physical memory and analyze information about the malware (which can then be used by the active security system to better secure the infrastructure).

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: General—*Security and Protection*

General Terms

Security, Design, Experimentation, Management

Keywords

Network security, central management, digital forensics

1. INTRODUCTION

The task of securing an enterprise's, nation's, or military's cyber infrastructure is increasingly complex due to the diversity in attack vectors, sources, and targets. Recent articles regarding attacks on defense contractors [34] and Universities [31] illustrate the constant threat and diversity of ways attackers are able to infiltrate and, in many cases, extract confidential data. Not only is it important to be able to detect attacks, we need to be able to perform attribution, and prevent future attacks. Unfortunately, today's systems lack the ability to be highly active and customized for context-aware individual responses. At best, today's security systems can detect attacks, block traffic, and perform post-incident response and investigation.

To address these complex threats, we propose *active security* which 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. Recent research has capitalized on the network programmability of software-defined networks to introduce some dynamic network security applications [35]. With active security, we believe we must go even further where, "how the network directs traffic", is only one aspect.

In this paper we first motivate active security by discussing some of the main security systems of today and an example where these systems fall short (Section 2). We then present the core components of active security that provides continuous monitoring and reaction to the network infrastructure (Section 3). The overall architecture of a platform for active security is then presented (Section 4). Following that, we detail our prototype implementation (Section 5) and a complete example of active security where a machine was infected, an intrusion detection system (IDS) detected anomalous egress traffic, triggered in-attack volatile memory forensics of the affected machine, automatically analyzed the memory contents to recover a copy of the malware executable and extract other open sockets for which anomalous traffic was not detected, and then reconfigure the network to block the traffic associated with these sockets (Section 6). We wrap up with a discussion of some challenges (Section 7) and conclusions (Section 8).

2. MOTIVATION

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

2.1 Today's Individual Components work Individually

Today's tools to secure network and computing infrastructures are a mixed collection of individual components which can be configured, but have limited active programmability.

- **Firewalls:** can block traffic based on the specified configuration. They are good at blocking traffic from the outside, but attackers often find holes in the firewall configuration and can often perform host-to-host attacks within the network – avoiding the firewall completely.

- **Intrusion detection/prevention systems**: are capable of monitoring traffic, but require many machines to handle a full traffic load. As such, traffic is often sampled, leading to only seeing 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.

- **Digital Forensics:** include tools and techniques to find and correlate information in the scattered logs throughout the entire infrastructure. However, operators cannot be sure that the correct information is being logged and whether or not the attacker already cleaned up.

- **Security Information and Event Management:** provide real-time aggregation of events and logged information. These are a step in the right direction, but still limited from a programmability standpoint.

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 what's being monitored or altering the IP address assignment to protect a current target). Strides have been made in the research community, such as FRESCO [35], which provides a scripting interface to enable the creation of security applications on an OpenFlow network. 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 with active security.

In industry, there has been point solutions to use a feedback loop to, for example, detect DDoS attack and reconfigure routers to block associated traffic [1]. The Cisco CS-MARS [9] and SolarWinds [3] 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.

2.2 Example: Active In-Attack Attribution

As a motivating example of something that cannot be achieved easily with today's systems, we consider the value of volatile memory for forensic evidence and even in helping to better defend against attacks. We'll revisit this example in more depth in Section 6 with a demonstration using our prototype.

In practice, volatile memory contents are often lost piece of information in intrusion investigations, yet would prove quite valuable. Consider the value of being able to see memory contents *during* an attack – that is, before the attacker has cleaned up. First, we can get a complete picture of the attack as many exploit payloads are able to reside completely in memory and clean themselves up after execution. By extracting system memory contents, we can obtain a copy of the executable, identify any open files and network sockets, etc. This can be very powerful in understanding the attack as well as attributing the attack to a person or organization. This can also help in defending the network – *e.g.,* the open sockets may be traffic that security middleboxes have not detected as anomalous, but through the forensics, the traffic can be dealt with.

Second, obtaining a memory dump during an active attack can enable forming a more complete and reliable assessment of the damage that was done. For example, systems today already have the ability to capture traffic traces. Yet, if the attacker was leaking sensitive information through encrypted channels, we may not know what was leaked. By grabbing the contents of system memory, we can gain access to cryptographic keys that would enable determining what was transmitted, which is particularly important in the case of intellectual property or sensitive government information.

This example only touches on one aspect of security that is not possible with today's systems. Through the software programmability offered by an *active security* system, we can enable these capabilities and others such as migrating malicious code to a quarantined system and reconfiguring the network to direct flows associated with the malware to the quarantined system so that we can monitor the malware behavior while cleaning up the live system. We will elaborate in the following sections.

3. ACTIVE SECURITY COMPONENTS

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. In this section we first outline requirements of such a system and then in the following section provide details of the proposed platform to meet each requirement. At a high-level, we are inspired by concepts from software-defined networking (SDN) 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.

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 – *e.g.,* using and configuring a firewall. Unfortunately, while configuration to protect an infrastructure is somewhat well understood today, it still leads to many errors [36]. Research has been performed in this space in how to realize less error-prone firewall configuration [12, 15]. This part of active security is not new, but it is an essential foundation upon which we may build a context-aware security system.

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 [35] 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.

3.3 Adjust: Alter the network to better defend or monitor

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 [25] has been positioned as a counter measure, and even demonstrated within an OpenFlow network [24]. 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 [22, 29] 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 [18]). Instead of statically defining, we need the dynamic ability to alter what we're 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 [37, 28]. Likewise, researchers have proposed mechanisms to ensure an 'accountable virtual machine' is behaving properly [23]. 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.

3.4 Collect: Actively perform 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 [32, 14]. 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.

3.5 Counter: Initiate reconnaissance and counter-attack

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're interacting with a real system when, in reality, it's 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 attacker's 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 – *e.g.,* launching a denial-of-service against the attacking machine.

4. PLATFORM FOR ACTIVE SECURITY

Shown in Figure 1 is the overall architecture of our active security system (we discuss the security of the framework itself in Section 7.1). Operators will run (potentially customized) software programs to specify the behavior of the active security. Further, operators will interact with the security controller through reports and direct control interfaces. The controller platform will provide APIs to the programmer and a plug-in ability to interface with different types of devices. This plug-in architecture includes interfaces to receive information from the sensors, perform forensic evidence collection, and configure the infrastructure – all on-demand at run-time. There will be an additional interface to more sophisticated actions, namely reconnaissance and counter-attack software. Each of these plug-ins interact with the various devices in the infrastructure.

5. 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, detailed further in Section 6. We envision a tight coupling with software-defined

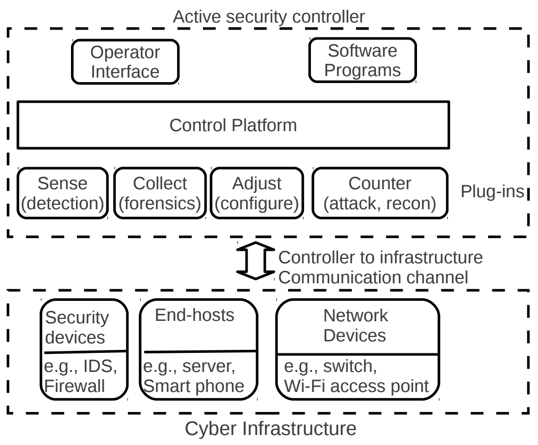

Figure 1: Platform for Active Security.

networking which provides the programmatic control over the network devices, and so we chose to integrate within the FloodLight [6] controller platform rather than run separately (a decision we can revisit in the future). We extended FloodLight to not only control the network, but to also interact with end-host systems and security middleboxes.

Interaction with an IDS security middlebox: Within our test implementation, we chose the Snort [8] Intrusion Detection System as the source of sensory input to our active security controller. We parse Snort's 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) [7] 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 later 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.

6. COMPLETE ACTIVE SECURITY EXAMPLE

As a complete walk through of an example of active security in practice, we expand upon the motivating example of in-attack volatile memory forensics. Importantly, we have built and demonstrated this scenario using our prototype (discussed in Section 5).

The scenario for this example is an enterprise network which uses good security practices, uses up-to-date anti-virus software, has ownership of the devices attached to the network, and can ensure a software configuration of the devices before allowing the devices onto the network (we discuss 'bring your own device' (BYOD) in Section 7.2).

Figure 2 illustrates a small-scale version of a generic network architecture which we used to realize our prototype. Two directed arrows, depict the path from attacker to victim, in red, and a purple arrow shows the return path from victim to attacker, illustrating the attack flow. Our network consists of a gateway router, which acts as a layer 3 device separating the internal network from the public Internet and wireless access points, a network Intrusion Detection System, two OpenFlow switches at the core and access layers, our active security controller (which is integrated with an SDN controller to control the network), an Internet proxy server, and some form of secure storage for evidence preservation and analysis. We will briefly walk through an attack scenario referencing Figure 1 and the associated time line detailed in Table 1.

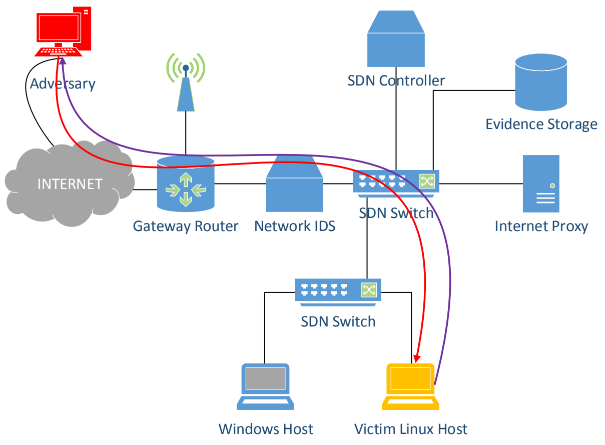

Figure 2: Network architecture for the active security example.

We start with the assumption that the attacker was able to penetrate the firewall to place an executable on the end host (*e.g.,* overwriting an existing application), evade the anti-virus software (*e.g.,* by performing an xor with a bitmask), and execute the software on the end host (*e.g.,* by the user running the executable). In particular, we achieved this through a client-side exploit, altered the calc.exe executable,

Table 1: Time of events of the example to obtain memory contents and reconfigure the network.

Time from start (sec)	Event
0.000	User Executes Malware on client system
0.763	Detection by IDS
1.769	Controller Receives alert/activate mempull FL module
62.483	Volatile Memory Image Complete
124.175	Volatile Memory Transferred to Storage
125.819	Volatility Recovers Network Sockets and Malware Executable

and were able to evade 31 out of 42 different anti-virus software packages we tested with this simple exploit (including some popular ones such as AVG, ClamAV, F-Secure, McAfee, and Sophos).

At time zero, the calc.exe application is executed, and the TCP socket is immediately established to "call home" and provide the attacker with a command shell. After less than a second, Snort IDS detects this anomalous egress traffic and about one second later, our active security controller is notified of the IDS alert. Our active security controller then initiates the volatile memory capture module on the host machine specified by the IDS alert. An SSH connection is immediately established and the LiME kernel module is loaded. At this point, LiME immediately images volatile memory, lasting approximately 60 seconds (for the entire 8GB of memory – in the future, we'll look at being more targeted in which memory to grab). Upon completion of the memory imaging, the image is securely transferred back, taking 62 seconds. Upon completion of the transfer, we scripted Volatility to parse several pieces of information from the memory capture relevant to this example. In particular, we extracted the decoded executable contents which we can then analyze offline with the Immunity debugger [2]. We also extracted a list of open TCP sockets. In this case, there was an additional TCP connection open that our IDS missed. While there are many possible things one might do with this information, we fed it into the FloodLight controller to simply block the traffic that evaded detection.

Upon detecting this anomalous traffic, the active security controller has successfully parsed the IDS alert for pertinent details, delivered this information to a central, active security controller, and triggered a programmed response to collect physical memory on the victim client. The key was successfully detecting anomalous behavior, successful analysis of the attack through capturing volatile memory in real-time, and programmatically analyzing the capture and altering the network based on its contents.

While we only looked for open network sockets, we also have the ability to scrape process trees, determine loaded and unloaded DLLs, inspect session IDs to determine the user level the attacker attempted to achieve, etc. A substantial amount of work has been done in the area of examining and correlating Virtual Address Descriptor (VAD) [19] values, loaded kernel modules, mutual-exclusive objects, and many other physical memory artifacts for the purpose of detecting root kits.

7. DISCUSSION

With our initial prototype to demonstrate the core architecture, and the working evaluation of a complete scenario, we have provided concrete examples of the possibilities that active security brings to help secure network and computing infrastructures. There, however, remain many open question. We discuss a few here.

7.1 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 [13], operating systems [26, 27, 20, 38], languages [16, 10, 30], and SDN controllers [17, 33]. 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.

7.2 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 [5]. Further, virtualization is being introduced as a means to allow employees to bring their own device, but employers to control the software environment [11, 4]. It might be possible to leverage that to be able to execute software on the mobile device without impacting the user's personal VM.

7.3 Sensory Input, Tools, and Techniques

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.

7.4 Beyond the Enterprise

While our example in Section 6 focused on an enterprise network, active security can extend beyond the scope of enterprise networks into the realms of datacenter networks and web application security. More and more enterprise infrastructure is moving to cloud infrastructures, and with it, many of the same security issues will remain.

One unique aspect of a cloud infrastructure is that the attackers may be using the same infrastructure as the victims [21]. The easy to access computing power of hosted cloud infrastructures has led many attackers to use the cloud for illicit activity. The challenge previously discussed of malware cleaning up after itself has a direct analogy that virtual machines can be spun up and shut down, and leave virtually no trace. We can envision a cloud provider introducing a security based API that can be triggered from tenants and access control only available at the provider level – *e.g.,* the provider is in position to grab a memory dump of a VM that one tenant suspected as malicious. Of course, this needs to be further studied to ensure proper privacy and protection against either putting too much load on the provider or interfere with the suspected malicious tenant until they are found to actually be acting maliciously.

8. CONCLUSION

In the face of ever increasing number and complexity of cyber-attacks, new tools are needed to overcome the limitations of the rigidity of existing security infrastructure. In this paper we presented a vision for active security which provides the programmatic control over an infrastructure to better deal with the changing threat landscape. Using our prototype, we were able to interface with an IDS system and upon an alert for anomalous traffic, we collected a dump of volatile memory from the affected end-host machine which we then analyzed for information about open files and sockets and information that can be used to detect root-kits installed by the malware. This is only the first step in providing an active security system.

9. REFERENCES

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

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

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

[4] VMware Horizon Mobile. http://www.vmware.com/products/desktop_virtualization/mobile/overview.html.

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

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

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

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

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

[10] D. S. Alexander and J. M. Smith. The Architecture of ALIEN. In *Proc. International Working Conference on Active Networks (IWAN)*, 1999.

[11] J. Andrus, C. Dall, A. V. Hof, O. Laadan, and J. Nieh. Cells: a virtual mobile smartphone architecture. In *Proc. Symposium on Operating Systems Principles (SOSP)*, 2011.

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

[13] 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.

[14] A. M. Azab, P. Ning, Z. Wang, X. Jiang, X. Zhang, and N. C. Skalsky. Hypersentry: enabling stealthy in-context measurement of hypervisor integrity. In *Proc. ACM conference on Computer and communications security (CCS)*, 2010.

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

[16] 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.

[17] M. Canini, D. Venzano, P. Peresini, D. Kostic, and J. Rexford. A NICE way to test OpenFlow applications. In *Proc. Network System Design and Implementation (NSDI)*, Apr. 2012.

[18] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker. Ethane: taking control of the enterprise. In *Proc. SIGCOMM*, 2007.

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

[20] P. Efstathopoulos, M. Krohn, S. VanDeBogart, C. Frey, D. Ziegler, E. Kohler, D. Mazières, F. Kaashoek, and R. Morris. Labels and event processes in the asbestos operating system. In *Proc. symposium on Operating systems principles (SOSP)*, 2005.

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

[22] A. Greenberg, G. Hjalmtysson, D. A. Maltz, A. Myers, J. Rexford, G. Xie, H. Yan, J. Zhan, and H. Zhang. A clean slate 4d approach to network control and management. *SIGCOMM Comput. Commun. Rev. (CCR)*, 35(5):41–54, Oct. 2005.

[23] A. Haeberlen, P. Aditya, R. Rodrigues, and P. Druschel. Accountable virtual machines. In *Proc. USENIX conference on Operating systems design and implementation (OSDI)*, 2010.

[24] J. H. Jafarian, E. Al-Shaer, and Q. 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.

[25] S. Jajodia, A. K. Ghosh, V. Swarup, C. Wang, and X. S. Wang, editors. *Moving Target Defense - Creating Asymmetric Uncertainty for Cyber Threats*, volume 54 of *Advances in Information Security*. Springer, 2011.

[26] G. Klein, K. Elphinstone, G. Heiser, J. Andronick, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, et al. seL4: Formal verification of an OS kernel. In *Proc. symposium on Operating systems principles (SOSP)*, 2009.

[27] M. Krohn, A. Yip, M. Brodsky, N. Cliffer, M. F. Kaashoek, E. Kohler, and R. Morris. Information flow control for standard os abstractions. In *Proc. symposium on Operating systems principles (SOSP)*, 2007.

[28] C.-C. Lin, M. Caesar, and J. V. der Merwe. Towards Interactive Debugging for ISP Networks. In *ACM Workshop on Hot Topics in Networks (HotNets)*, Oct. 2009.

[29] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. OpenFlow: Enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev. (CCR)*, 38(2), 2008.

[30] S. Nagarakatte, J. Zhao, M. M. Martin, and S. Zdancewic. SoftBound: highly compatible and complete spatial memory safety for c. In *Proc. conference on Programming language design and implementation (PLDI)*, 2009.

[31] R. 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.

[32] N. L. Petroni, Jr., T. Fraser, J. Molina, and W. A. Arbaugh. Copilot - a coprocessor-based kernel runtime integrity monitor. In *Proc. USENIX Security Symposium*, 2004.

[33] P. Porras, S. Shin, V. Yegneswaran, M. Fong, M. Tyson, and G. Gu. A security enforcement kernel for OpenFlow networks. In *Proc. workshop on Hot topics in software defined networks (HotSDN)*, 2012.

[34] M. Riley and B. 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.

[35] S. Shin, P. Porras, V. Yegneswaran, M. Fong, G. Gu, and M. Tyson. Fresco: Modular composable security services for software-defined networks. In *Proc. Network and Distributed System Security Symposium (NDSS)*, February 2013.

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

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

[38] N. Zeldovich, S. Boyd-Wickizer, E. Kohler, and D. Mazières. Making information flow explicit in histar. In *Proc. symposium on Operating systems design and implementation (OSDI)*, 2006.