Resonance: Dynamic Access Control for Enterprise Networks

Ankur Nayak, Alex Reimers, Nick Feamster, Russ Clark
School of Computer Science, Georgia Tech

ABSTRACT

Enterprise network security is typically reactive, and it relies heavily on host security and middleboxes. This approach creates complicated interactions between protocols and systems that can cause incorrect behavior and slow response to attacks. We argue that imbuing the network layer with mechanisms for dynamic access control can remedy these ills. We propose *Resonance*, a system for securing enterprise networks, where the network elements themselves enforce dynamic access control policies based on both flow-level information and real-time alerts. Resonance uses programmable switches to manipulate traffic at lower layers; these switches take actions (*e.g.*, dropping or redirecting traffic) to enforce high-level security policies based on input from both higher-level security policies and distributed monitoring and inference systems. We describe the design of Resonance, apply it to Georgia Tech's network access control system, show how it can both overcome the current shortcomings and provide new security functions, describe our proposed deployment, and discuss open research questions.

Categories and Subject Descriptors: C.2.1 [Computer-Communication Networks]: Network Architecture and Design C.2.6 [Computer-Communication Networks]: Internetworking

General Terms: Algorithms, Design, Security

Keywords: enterprise networks, access control, programmable networks

1. Introduction

Enterprise networks host many heterogeneous and potentially untrusted devices that may be vulnerable to compromise. Despite significant advances in host security, the growing number and types of network devices—ranging from desktops to laptops to handhelds to media consoles—makes it increasingly difficult to secure every device that connects to the network. These devices run a variety of operating systems and are subject to a diverse set of vulnerabilities. In the face of these challenges, the network must authenticate these new devices as they arrive *and* monitor their behavior to detect violations of various security policies (*e.g.*, the presence of unauthorized or compromised hosts).

Today, authenticating and securing hosts on enterprise networks is challenging, and network operators typically rely on a cocktail of reactive, ad hoc techniques. Operators implement policy using middleboxes, intrusion detection systems, and a collection of complex network configurations. This *post hoc* approach creates a plethora of independent, difficult-to-manage devices that interact in unexpected ways, resulting in weaker security, incorrect operation (e.g., misconfiguration [6]), or both. The interaction between these many "moving parts" creates a system that is brittle and unresponsive in the face of various security threats. For example, access control on the Georgia Tech campus network entails interaction between firewalls, dynamic address (DHCP) servers, virtual LANs, intrusion detection systems, and the switches and routers themselves [16]. Enterprise network security today also places a considerable—perhaps unreasonable—burden on both users and network operators to ensure that hosts are patched and kept up-to-date, running virus scanners and host-level intrusion detection systems, etc.

Instead of placing trust in the end hosts or relying on security middleboxes, an enterprise network should offer mechanisms that directly control network traffic according to dynamic, fine-grained security policies, and in response to input from distributed network monitors. Extending the metaphor of a network operating system [10] to the design of *secure* networks, we present the design of *Resonance*, which provides mechanisms for directly implementing dynamic network security policies in the network, at devices and switches, leaving little responsibility to either the hosts or higher layers of the network. We draw inspiration from the design of secure operating systems, where complex system components are built using, small, hardened, trusted components as a base. Similarly, Resonance imbues the network layer itself with the basic functions needed to implement security policies, as well as a control interface that allows monitoring systems to control traffic according to predefined policies.

As in previous work (*e.g.*, Ethane [1]), Resonance controls traffic using policies that a controller installs in programmable switches. Extending this paradigm, we create a *dynamic access control* framework that integrates the controller with monitoring subsystems. This integration allows an operator to specify how the network should control traffic on an enterprise as network conditions change. For example, Resonance can automatically quarantine hosts or subsets of traffic when a compromise or other security breach is de-

WREN'09, August 21, 2009, Barcelona, Spain.

tected. In this paper, we explore this design in the context of access control and monitoring on the Georgia Tech campus network. To the best of our knowledge, no campus network today allows dynamic, fine-grained network policies based on integration with monitoring systems. We explore how Resonance not only simplifies the implementation of network security policies, but also enables fine-grained security policies and a wider range of features.

Recent trends enable this integration of dynamic monitoring and control. First, *programmable (and software-based) network devices* [3, 14] allow more direct, fine-grained control over network traffic. At first blush, programmable network devices might seem to present yet another source of complexity, but we believe that this programmability actually presents an opportunity to proactively secure the network layer. Second, *distributed network monitoring algorithms* can now quickly and accurately correlate traffic from many distinct (and often distributed) sources to detect coordinated attacks (*e.g.*, for detecting botnets [9] and spammers [15]). Finally, the trend towards logically centralized network control [7, 8] allows us to more easily integrate distributed network monitoring with dynamic network control.

Consider the task of quarantining an infected host, which involves specifying a security policy (*e.g.*, which indicates the nature of traffic that might indicate a compromised host), monitoring the network traffic to detect possible violations of security policy, and taking the appropriate action to correct the violation. This task currently requires network administrators to (1) install on-path firewalls that perform on-path inspection of traffic; and (2) update firewall rules when a compromised host is detected. Instead, Resonance provides an interface for distributed inference algorithms to directly control the behavior of network traffic. Distributed inference using existing subsystems can monitor traffic at higher layers and detect compromised hosts (*e.g.*, [9, 15]). Resonance can integrate these alarms with *actions* that the switches directly implement (*e.g.*, redirecting, rate-limiting, or dropping traffic).

Despite the promise of Resonance's design and recent trends that could make its deployment more feasible, we are grappling with many challenges in our initial test deployment. First, Resonance must *scale* to a large number of users and traffic flows: The residential network alone on the Georgia Tech network has 16,000 active users. The system must provide flexibility and dynamic control, without storing a prohibitive amount of state at the switches themselves or introducing excessive delays on packet forwarding. Second, Resonance must be *responsive* to various changes in network policy: It must quickly authenticate legitimate network hosts and devices, and it must quickly quarantine hosts that violate security policies. Third, the controller and programmable switches must be *integrated with real-time monitoring* and alert systems; the controller must be able to quickly correlate and synthesize alerts and quickly send control messages to switches to affect traffic flows. Finally, the control channel must be secure: the controller and switch interfaces must be robust to attack, and the control channel between the controller and the devices must be available.

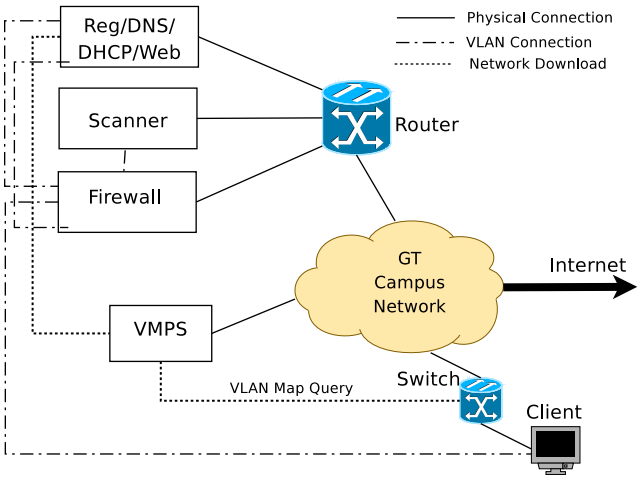

Figure 1: Current START Architecture.

This paper makes the following contributions. First, we describe the architecture for dynamic network monitoring and access control in the Georgia Tech campus network and enumerate various shortcomings of this design. Second, we describe Resonance, a framework for implementing dynamic, fine-grained access control in enterprise networks. As part of this design, we introduce a new framework and set of mechanisms for specifying and implementing dynamic access control in enterprise networks. Third, we explore how Resonance can simplify and improve dynamic access control in the context of the Georgia Tech campus network. Finally, we describe research challenges and an initial testbed where we plan to evaluate Resonance before its ultimate deployment on the campus network.

The rest of the paper is organized as follows. Section 2 presents an overview of the current authentication infrastructure on the Georgia Tech campus network and background on OpenFlow. Section 3 describes the Resonance design; Section 4 describes how we plan to apply Resonance to access control on the Georgia Tech network. Section 5 describes challenges, Section 6 presents related work, and Section 7 concludes with a summary and research possibilities.

2. Background

We describe the network access control problem in the context of the Georgia Tech campus network and introduce OpenFlow [14], an interface for programmable switches on which we base our design.

2.1 Access Control and Monitoring

Campus and enterprise networks are often large, heterogeneous, and unmanageable. Thus, network administration can be both troublesome, manual, and error-prone. Network administrators often encounter situations where machines are infected or compromised. Today, the network operator must manually remove or quarantine the machine from the network, which is tedious. The network should offer flexible, fast control over network traffic while also scaling to a large number of users and traffic flows. To the extent possible,

network management should also be automated, to ease the burden on the network administrators.

2.1.1 Current system overview

Figure 1 shows the current START architecture [16], the authentication system deployed in the Georgia Tech Campus. It is currently based on virtual LANs (VLANs) and VLAN Management Policy Server (VMPS) [19]. The START system supports the following functions:

Registration The registration system provides the Web interface to the backend registration database, DHCP, DNS, authentication, and updates for external systems. The Web interface guides users through the registration process. The DNS server returns the IP address for the registration server for all DNS queries, except for a list of domains needed for patching (*e.g.*, windowsupdate.com). The system runs two DHCP servers: One for the unregistered VLAN, and one for the registered VLAN. Each instance has its own configuration files that are created automatically from data in the registration system's database.

Scanning During the registration process, systems are scanned for known vulnerabilities. If the scan reveals vulnerabilities, the user is presented with these vulnerabilities and given an opportunity to update the system. The firewall allows traffic to the appropriate update servers.

Firewall The registration VLAN uses a firewall to block network traffic to unregistered hosts. The firewall allows Web and secure Web (*i.e.*, port 80 and 443) traffic to pass so that hosts can reach update sites. Various routers and switches create the necessary VLANs. The local switches determine the VLAN for each machine that joins the network. The switch will download VLAN maps periodically from a VMPS. Unknown MAC addresses are assigned to the unregistered VLAN and known MAC addresses are placed onto the appropriate subnet. VMPS periodically downloads the VLAN maps from the registration server. Security is enforced with ARP tables that map each MAC address to its registered IP address.

2.1.2 Problems with the current design

The current architecture has several shortcomings:

1. **Access control is too coarse-grained.** START deploys two different VLANs to separate infected or compromised machines from healthy machines. This segregation results in all compromised hosts residing on a single VLAN; such a configuration does not provide proper isolation, since these infected hosts are not isolated from each other. Additionally, relying on VLANs makes the system inflexible and less configurable, because VLANs typically map hosts to network segments according to MAC address, *not* according to individual flows.
2. **Hosts cannot be dynamically remapped to different portions of the network.** In the current configuration, when a machine is mapped to a different part of the network, it must be rebooted to ensure that it receives a public IP address, which is inconvenient because it relies on user intervention.
3. **Monitoring is not continuous.** Authentication and scanning only occur when a network device is initially introduced; if the device is subsequently compromised (or otherwise becomes the source of unwanted traffic), it cannot be dynamically remapped to the garden-walled portion of the network.

Many of the current shortcomings result from the fact that security functions have been added on top of the existing network infrastructure. This design was natural when switches needed to be treated as "black boxes"; however, switch vendors have begun to expose a standard interface, OpenFlow [14], whereby an external controller can affect how a switch forwards traffic. We summarize OpenFlow below.

2.2 OpenFlow

OpenFlow-enabled switches expose an open protocol for programming the flow table and taking actions based on entries in these flow tables. The basic architecture consists of a *switch*, a centralized *controller*, and end hosts. The switch and the controller communicate over a secure channel using the OpenFlow control protocol [14], which can affect flow table entries on the switch. Currently, all OpenFlow switches support three actions: (1) **Forward** this flow's packets to a given port or ports. This function allows packets to be forwarded. (2) **Encapsulate** and forward this flow's packets to a controller. In this case, the packet is delivered to a secure channel, where it is encapsulated and sent to a controller. This function may be used for the first packet in a flow, so a controller can decide if the flow should be added to the flow table. (3) **Drop** this flow's packets. Sections 3 and 4 explain how we use OpenFlow in the context of Resonance.

3. Resonance Design

We describe the design of Resonance, explain how this design enables fine-grained, dynamic control over traffic for implementing security policies.

3.1 Overview

The high-level Resonance architecture has the following salient features: a policy specification framework, distributed network monitoring, and the ability to take specific actions using programmable switches.

Policy specification framework We base our policy specification framework on existing access control frameworks; existing frameworks typically assign each principal to a security class. Because access control in Resonance is dynamic, each principal (*i.e.*, host) has both a security class and a state. Resonance allows a network operator to specify a variety of functions based on a principal's security class and state. For example, given information about a machine entering a compromised state, the controller can instruct switches to treat network traffic accordingly, based on a higher-level specification of security policy. In this paper, we focus on the access control framework, rather than the language to specify

the policy itself; ultimately, policy languages such as FSL might be extended to support our policy framework [12].

Resonance's policy specification framework provides significantly finer-grained access control than existing mechanisms. Today, hosts that belong to the same security group belong to a common VLAN, but all traffic on the same VLAN is subject to the same policy, and hosts on the same VLAN are not protected from each other. Additionally, because the VLAN identifier is only 12 bits, a network is restricted to 4,096 VLANs. These VLANs can be quickly exhausted on a large network with many fine-grained policies, and some policies—such as partitioning hosts from one another—are simply not feasible.

Distributed network monitoring Rather than relying on host-level security, Resonance leaves the task of detecting security violations to the network itself. Recent work has demonstrated that distributed inference based on analysis of network traffic can perform essential network management tasks, such as detecting compromised hosts [9], filtering spam [15], and troubleshooting network performance degradations [5, 17]. Resonance makes this monitoring intrinsic to the architecture: Network elements can forward reports about network traffic (*e.g.*, flow level information about DNS lookups or specific attack traffic) to a centralized location for performing this inference.

Most existing monitoring systems raise alerts based on the behavior, performance, etc. of individual hosts. These detection systems miss the opportunity to detect coordinated behavior that may suggest various events (*e.g.*, malware spreading, hosts participating in a botnet). By incorporating input from distributed inference systems, Resonance may be able to defend against a larger class of attacks.

Fine-grained, dynamic control with programmable switches Whereas today's networks completely decouple monitoring from lower-layer traffic control, Resonance allows switches to dynamically re-map clients based on other input (*e.g.*, alarms from distributed network monitoring systems, such as BotMiner [9] and SpamTracker [15]). Alert systems control traffic by sending messages to the controller, which in turn controls switch behavior via the standard, narrow OpenFlow-based switch interface. This refactoring keeps on-path forwarding decisions simple, while still allowing complex policies to be implemented through a standard control interface.

Resonance's coupling of distributed inference-based alert systems with programmable switches enables *dynamic* access control, whereby network devices may treat traffic differently based on the controller's view of a host's current state and security class. This ability to both specify and implement dynamic access control contrasts sharply with existing network configurations, whereby hosts mapped to a single VLAN, and this mapping is only overridden with manual operator intervention.

3.2 Policy Specification

In contrast to existing access control frameworks, Resonance's access control allows for *dynamic* policies. Dynamic policies are essentially lattice-based access control [4], except that each principal has both a security class *and a state*, where the state can change over time according to a set of transitions defined by the policy. The policy effectively dictates what actions a switch should take on traffic to and from a host that is of a particular security class and state. Dynamic policies allow the network switches to change how they control a host's traffic as network conditions change.

A principal's security class dictates the type and nature of access that the host has to other resources on the network and the extent to which the host is monitored. Based on the principal, a network operator may both restrict the type of access that a host has to the network and mandate that the host be monitored more or less extensively. For example, on a campus network, an operator might wish to ensure that low-privilege campus guests can only send traffic only to the global Internet (*e.g.*, to protect hosts on the enterprise from scans or attacks from a potentially untrusted host). Resonance facilitates such policies through a policy specification framework that determines: (1) the possible classes and states for each principal; (2) corresponding access control policies; (3) actions that network elements should take to enforce the policy; (4) a specification of how hosts can transition from one state to another.

Security classes As in traditional access control models [4], principals in a Resonance network have security classes that dictate the access that they (and their traffic) have to other resources on the network. Ultimately, we may extend Resonance so that every resource (*i.e.*, device connected to a switch port) has a security class, and the lattice will dictate how traffic may flow to one resource or another.

States and transitions Each security class has a pre-defined set of states (and a corresponding state machine) that determines what states a principal of that class can be in, what transitions are allowed, and what causes transitions between states. Each state may also have policies/actions that are to be taken for certain subpopulations of traffic (where traffic types are identified by flow attributes). Transitions can specify, for example, that different alerts from distributed monitoring systems can cause a host to transition from one state to another.

Actions Resonance switches use flow tables that have rules for matching traffic flows to actions; actions correspond to those in Section 2.2. This aspect of the design draws on the features that OpenFlow-based switches provide. The key extension is that switches may use multiple tables for any given principal, where the table that the switch uses at any time depends on the security class and the current state of that principal. Dynamic updates to tables could be implemented either by having the controller install new tables on the fly, or by storing multiple tables for each principal and swapping tables upon instruction from the controller.

In this paper, we assume that all resources and principals belong to a single security class. We focus on how to declare states, transitions, and actions, and how we can implement a

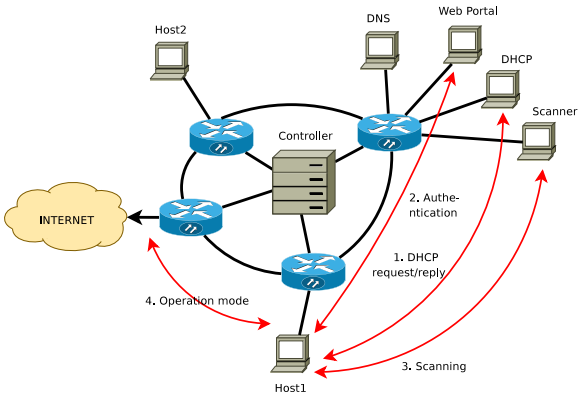

Figure 2: Applying Resonance to START.

dynamic access control framework by using policies to help the controller map hosts from one state to another. We leave the integration of security classes and lattice-based access control for future work.

3.3 From Specification to Operation

The controller implements the access control policies by installing the appropriate flow table entries into the switches themselves. Resonance uses the MAC address corresponding to a host's interface to map traffic back to the appropriate principal.[1] All specifications must have a start state, so that traffic from an unknown MAC address can be treated appropriately. Subsequently, the controller installs flow table entries into switches based on the security class and state of each MAC address. The controller then listens for updates about the state and security class of each host; when a host transitions to a different state (*e.g.*, if a host compromise is detected), the controller changes policies at the switches according to the specification language. The controller essentially "compiles" the dynamic access control specification into switch configurations. We envision the controller directly configuring the switch using OpenFlow, but other methods (*e.g.*, reconfiguration) are feasible.

4. Applying Resonance to START

In this section, we describe the application of Resonance to dynamic access control and monitoring in the Georgia Tech campus network. To simplify the initial design, we assume that all hosts are in the same security class, and that only their states are changing in response to results from authentication and scanning. In the future, we plan to explore how Resonance can also support more fine-grained access control by assigning specific security classes to both principals and switch ports.

Overview Figure 2 provides an overview of the network operation we aim to implement in Resonance. A device broadcasts a DHCP "discover" message. The DHCP server sends back a public IP address to the machine. To gain access to the wide-area Internet, the machine must authenticate itself via the START Web service; OpenFlow-enabled switches can redirect all HTTP requests from unauthenticated machines to the START Web site by default. Once a user authenticates the machine, the Web service saves the MAC address of the machine and updates flow-table entries to allow access to a restricted set of destinations (*e.g.*, Microsoft Update). At this point, a scanner examines the device for potential infections. If the machine passes the scan, the START Web service sends a request to controller to permit traffic from this machine to be forwarded to any destination. The network performs continual scanning of hosts, using distributed inference techniques (*e.g.*, SNARE [11], BotMiner [9]), quarantining them if necessary.

[1]This approach depends, of course, on MAC addresses not being spoofed, which is generally preventable in enterprise networks. We refer the reader to previous work [1] for a more detailed discussion of this issue.

Registration State	
Match	**Action**
Ethernet Type ARP (0x806)	FLOOD
UDP src_port=68 dst_port=67 (DHCP ports)	FLOOD
UDP src_port=67 dst_port=68 (DHCP ports)	FLOOD
TCP Dst port 80/443/8080	REDIRECT (to web portal: 192.168.1.3)
*	DROP
Operation State	
Match	**Action**
*	FORWARD/DROP (according to policy lookup)

Table 1: Flow table entries for Registration and Operation states.

4.1 States and Actions

The controller maintains a state machine for every MAC address connected to the network. Here, we will describe the policies for each state. Every machine can reside in one of four states. A host can be in the *Registration* state, the *Authenticated* state, the *Operation* state, or the *Quarantined* state. In the Registration state, the machine is in the process of authenticating itself. The Authenticated state enables the scanning software to scan for vulnerabilities in the host. In the Quarantined state, the machine is only allowed to access sites for downloading patches and updates; quarantined hosts cannot exchange traffic with each other. In the Operation state, switches forward traffic according to declared access control policies and enables proactive monitoring to detect infected machines in the network.

The controller manages the state of each machine and updates the flow table entries in the switches corresponding to the current states of the machine. Tables 1 and 2 enumerate the policies associated with each state of a host machine that is trying to authenticate and gain access to the network. For simplicity, we assume the following IP addresses for the various components: Resnet Router (192.168.1.1), DHCP Server (192.168.1.2), START Registration Web Service (192.168.1.3), vulnerability scanner (192.168.1.4), and START Quarantine Web page (192.168.1.5). We explain these policies in detail below. Of course, other entities on the network (*i.e.*, DNS server, DHCP server, Web portal) also need flow table entries; since these are less likely to change

Authenticated State	
Match	**Action**
Ethernet Type ARP (0x806)	FLOOD
Dst_IP=192.168.1.4 (Scanner's IP)	FORWARD
Src_IP=192.168.1.4 (Scanner's IP)	FORWARD
TCP dst_IP=update sites	FORWARD
TCP src_IP=update sites	FORWARD
UDP dst_port=53 (DNS port)	FORWARD
UDP src_port=53 (DNS port)	FORWARD
*	DROP
Quarantined State	
Match	**Action**
TCP dst_port=80/443/8080	REDIRECT (to quarantine web page)
*	DROP

Table 2: Flow table entries for Authenticated and Quarantined states.

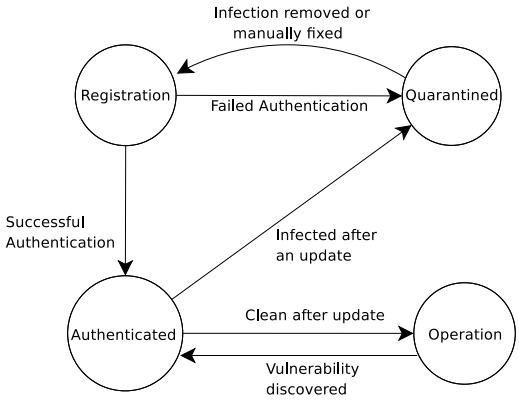

Figure 3: State transitions for a host. The controller tracks the state of each host and updates the current state according to inputs from external sources (*e.g.*, network monitors).

state, we focus our discussion on the aspects of Resonance that involve host authentication.

Registration state When the switch receives a packet from a machine for which it has no flow table entry, it forwards the packet to the controller. The controller maintains a database of authenticated machines, as well as the flow-table entries associated with them. Table 1 summarizes the flow policies that reside on the switch: (1) drop all packets other than HTTP, DHCP, and ARP; (2) broadcast all DHCP and ARP packets, and (3) forward HTTP requests to the Web portal.

Authenticated state In the Authenticated state, a host has been authenticated to the network and assigned to the appropriate security class, but it has not been verified to be free of infection. The host must be able to communicate freely with the scanner. The host is subject to all rules from the Registration state and is also allowed to communicate with the scanner. Table 2 summarizes these rules.

Operation state In addition to the normal forwarding policies, the controller also receives updates from network monitors about the IP addresses of infected (or otherwise misbehaving) hosts. The switch forwards all packets for this host according to the security classes and access control policies of each host under normal operation. If the controller receives an update about an infected host, it moves that host to the Quarantined or Registration state and updates switch flow tables accordingly; we discuss this operation in more detail in Section 4.2. Table 1 summarizes these rules.

Quarantined state The host is essentially disconnected from the network, except for the ability to retrieve patches from pre-specified Web sites. The switch drops all packets coming from the machine and redirects all HTTP requests to a Web page informing the user of the infection and provides pointers to sites where patches are available. Table 2 summarizes these rules.

4.2 Transitions

A host is initially in the *registration* state, at which point the switch forwards DHCP and ARP broadcasts. All other traffic from the client except for DNS and HTTP traffic to the portal is blocked, and all requests are redirected to the portal using DNS. If the client authenticates to the portal, the portal sends a message to the controller to move the host into the *authenticated* state ("successful authentication"); then, the controller updates flow table entries in the switches and triggers a scan of the host. If the client passes the scan, the scanner informs the controller to move the client into the *operation* state ("clean after update"). Otherwise, the client is moved to the *quarantined* state. In both cases, the controller updates the flow tables accordingly.

4.3 Resonance Step-by-Step

In this section, we explain how Resonance works step-by-step when a host connects to the network. We also describe how the controller manages the flow-table entries when two hosts attempt to communicate. Finally, we explain how a machine changes states and how the controller changes the flow-table entries accordingly.

Let us consider a simple setup with four OpenFlow switches, one controller, two hosts, and four servers, as shown in Figure 2. OpenFlow switches establish a connection with the controller using a secure channel. When a new host is introduced on the network, it first broadcasts a DHCP discover message; when the first DHCP packet is received by the switch connected to the host, it sends this packet to the controller over a secure channel. According to the policies in Table 1, the controller (1) establishes a flow-table entry to allow DHCP and ARP communication with the host; (2) adds the host to its database of hosts and marks its state as "Registration".

In the Registration state, if a host initiates any traffic that is not DHCP or ARP, the controller installs a new flow-table entry into the switch with action="DROP", unless the traffic is HTTP, in which case the controller installs an entry to redirect traffic to the portal, which redirects the user to the authentication Web site. A machine in the Registration or Quarantined state cannot initiate a connection, but it can always receive packets from a machine in the Operation state. We make this policy rule for simplicity.[2]

[2]An immediate implication is that a communication is possible from Operation state machines to Quarantined host machines. Because Quarantined

The Web portal allows the user to authenticate and notifies the controller of the status of the authentication via a separate connection. Upon successful authentication, the controller moves the host to Authenticated or Quarantined state. It then deletes all old flow-table entries corresponding to the host's MAC address and installs a new set of flow-table entries, as shown in Table 2. The only change made from Registration state to Authenticated state is that the host can communicate with the scanner and update sites. The scanner scans the machine for potential vulnerabilities. If the machine is found to be vulnerable, it is redirected to update sites to patch potential vulnerabilities. Once the update patches have been applied, the scanner notifies the controller, which then transfers the host to the Operation state and updates the flow-table entries accordingly.

Once in the Operation state, the host can connect to any other Internet destination. During normal operation, a host may become compromised. If network alarms inform the controller about the event, the controller can then shift the host to Authenticated state.

5. Challenges

Scale When deploying the architecture on the campus network, we expect to encounter numerous challenges involving scalability. For example, the Georgia Tech residential network must support approximately 16,000 users; the portion of the campus that runs START comprises more than 13,000 network ports, and future plans include expanding START to more than 40,000 active ports across academic buildings and merging START with the (separate) access control system currently used for the campus wireless network. A significant challenge will be implementing dynamic, fine-grained policies with flow-table entries, without exhausting switch memory or slowing forwarding. Recent proposals for optimizing customizable forwarding [2] may offer a useful starting point.

Responsiveness End hosts and network devices must be able to quickly authenticate to the network controller; similarly, the network must be able to quickly quarantine a compromised host and curtail unwanted traffic. The current design is inadequate in this regard, as it has a single VLAN for quarantined hosts and requires hosts to re-boot to reassign hosts from one VLAN to another. The Resonance architecture offers more fine-grained, dynamic control over hosts' traffic, but the control framework between the switches and controller must still be able to map hosts from one part of the network to another as quickly as possible. To enable this responsiveness, distributed inference must be fast, and the controller must be able to quickly and reliably alter the behavior of the switches themselves.

Integration with monitoring The current START network access control system scans hosts when they are first introduced into the network but cannot re-assign these hosts to different networks when they are deemed to be compromised. In our ongoing work, we will integrate alarms that arise from distributed monitoring and inference into mechanisms that can affect traffic flows more directly.

machines cannot initiate external communication, they are relatively immune to threats.

Securing the control framework The effectiveness of Resonance depends on the existence of a secure, reliable channel between the controller and the switches. The control messages between the controller and the switches must be authenticated (so that switches do not alter their behavior based on arbitrary control messages), and the channel must remain reliable and available, even when network utilization is high or the network itself comes under attack.

6. Related Work

Resonance draws inspiration from 4D [8] and Ethane [1], both of which advocate controlling network switches from a separate, logically centralized system. Ethane [1] is perhaps the most closely related work to Resonance. Like Ethane, Resonance defers traffic control to a centralized controller; however, Ethane does not support continuous monitoring and inference-based policy control. Ethane focuses primarily on host authentication, as opposed to security related problems such as monitoring and containment. Resonance extends the Ethane paradigm by exploring how *dynamic* security policies and actions (*e.g.*, actions based on alerts from distributed detection systems) could be more directly integrated into the network fabric.

NOX is a recently proposed "network operating system" that provides a uniform, centralized programmatic interface for a network [10]. Whereas NOX provides a platform for experimentation and research with new protocols, our proposed architecture should improve security by embedding security into the network itself. NOX could serve as a platform which we could use as the basis for our architecture. FSL is a policy language for NOX that allows network operators to write and maintain policies efficiently. Resonance focuses more on creating the actual policies that relate to dynamic access control and monitoring in enterprise networks

Resonance allows network devices to operate on the granularity of flows. This function is enabled by the emerging OpenFlow standard [14] and has origins in the designs of earlier protocols (*e.g.*, ATM [13]) and programmable switch architectures [18]. Recent trends in packet forwarding architectures (*e.g.*, [2]) have tried to achieve a similar shift towards the lower layers by having the software part of the switch make forwarding and pass it on to the hardware.

Some of the features of Resonance can be implemented using today's protocols. VMPS [19] allows a network to map a host to its corresponding VLAN based on its MAC address. However, network operators achieve this mapping via manual configuration; if a host needs to be re-mapped based on a change in its state (*e.g.*, if the host becomes compromised), VMPS provides no mechanism for automatically remapping such a host; this remapping must either be done manually, or a higher-layer, on-path security middlebox must take appropriate action. Resonance's access control is both more dynamic and more fine-grained than the access control enabled by VMPS and VLANs.

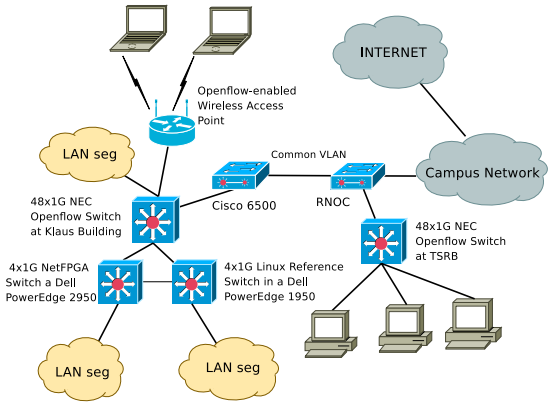

Figure 4: Research testbed.

7. Summary and Future Work

Existing enterprise networks leave network monitoring and access control to higher layers (*e.g.*, DHCP, application-level intrusion detection, etc.) and place considerable amounts of trust and responsibility into the network devices themselves, resulting in complex, error-prone configurations for enforcing security policies. To remedy these ills, network access control must be more dynamic and fine-grained, and it must make as few assumptions as possible about the behavior of the host. We have introduced a new framework, Resonance, for specifying dynamic access control policies for networks, described how this might be implemented in an OpenFlow-based architecture, and shown how to apply Resonance in the context of the Georgia Tech's network access control framework. In addition to the test and operational deployments themselves, we are exploring how Resonance can support more complex access control policies.

Testing and deployment Figure 4 illustrates the initial deployment platform that we are building to test the OpenFlow-based START architecture. The deployment is a dedicated network that is physically separate from the production network and yet has its own IP prefix and upstream connectivity. This platform will allow us to deploy and test Resonance before deploying it on the production network.

The campus network currently supported by START was recently upgraded to include approximately 275 switches that are capable of supporting the OpenFlow firmware. One of the more significant practical challenges in the campus deployment will be straining the scalability of the system on a production network without disrupting connectivity. For example, the proposed architecture may involve installing many flow table entries in the switches, which may either exhaust memory or slow lookup performance if entries are not stored efficiently, or if state is not offloaded to the controller. To address this concern, we will first stress-test the design on the research testbed and subsequently the architecture on a smaller number of production switches before completely rolling out the architecture.

Acknowledgments

This work was funded by NSF CAREER Award CNS-0643974 and NSF Awards CNS-0721581 and CNS-0751134. We thank David Andersen, Ron Hutchins, Hyojoon Kim, Richard Mortier, Jennifer Rexford, and Matt Sanders for helpful feedback and suggestions.

REFERENCES

[1] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker. Ethane : Taking control of the enterprise. In *SIGCOMM '07*, 2007.

[2] M. Casado, T. Koponen, D. Moon, and S. Shenker. Rethinking packet forwarding hardware. In *Proc. Seventh ACM SIGCOMM HotNets Workshop*, Nov. 2008.

[3] Cisco Application eXtension Platform Overview. `http://www.cisco.com/en/US/prod/collateral/routers/ps9701/white_paper_c11_459082.html`.

[4] D. E. Denning. A lattice model of secure information flow. *Communications of the ACM*, 19(5):236–243, May 1976.

[5] N. Duffield. Simple Network Performance tomography. In *Proc. ACM SIGCOMM Internet Measurement Conference*, Miami, FL, Oct. 2003.

[6] N. Feamster and H. Balakrishnan. Detecting BGP Configuration Faults with Static Analysis. In *Proc. 2nd Symposium on Networked Systems Design and Implementation*, Boston, MA, May 2005.

[7] N. Feamster, H. Balakrishnan, J. Rexford, A. Shaikh, and K. van der Merwe. The case for separating routing from routers. In *ACM SIGCOMM Workshop on Future Directions in Network Architecture*, Portland, OR, Sept. 2004.

[8] 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. *ACM Computer Communications Review*, 35(5):41–54, 2005.

[9] G. Gu, R. Perdisci, J. Zhang, and W. Lee. BotMiner: Clustering analysis of network traffic for protocol- and structure-independent botnet detection. In *17th USENIX Security Symposium*, 2008.

[10] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker. NOX: towards an operating system for networks. *ACM SIGCOMM Computer Communication Review*, 38(3):105–110, July 2008.

[11] S. Hao, N. Syed, N. Feamster, A. Gray, and S. Krasser. Detecting Spammers with SNARE: Spatio-temporal Network-level Automatic Reputation Engine. `http://www.cc.gatech.edu/~feamster/papers/snare-tr.pdf`, Feb. 2009. In submission; Earlier version appeared as Georgia Tech Technical Report GT-CSE-08-02.

[12] T. Hinrichs, N. Gude, M. Casado, J. Mitchell, and S. Shenker. Expressing and enforcing flow-based network security policies. Technical report, Dec. 2008.

[13] P. Newman, G. Minshall, and T. L. Lyon. IP Switching - ATM under IP. *IEEE/ACM Trans. Netw.*, 6(2):117–129, 1998.

[14] OpenFlow Switch Consortium. `http://www.openflowswitch.org/`, 2008.

[15] A. Ramachandran, N. Feamster, and S. Vempala. Filtering spam with behavioral blacklisting. In *Proc. 14th ACM Conference on Computer and Communications Security*, Alexandria, VA, Oct. 2007.

[16] Scanning Technology for Automated Registration, Repair and Response Tasks. `https://start.gatech.edu/`.

[17] M. B. Tariq, M. Motiwala, and N. Feamster. NANO: Network Access Neutrality Observatory. In *Proc. 7th ACM Workshop on Hot Topics in Networks (Hotnets-VII)*, Calgary, Alberta. Canada., Oct. 2008.

[18] J. van der Merwe, S. Rooney, I. Leslie, and S. Crosby. The Tempest - A Practical Framework for Network Programmability. *IEEE Network*, 12(3):20–28, May 1998.

[19] Configuring Dynamic Port VLAN Membership with VMPS. `http://www.cisco.com/univercd/cc/td/doc/product/lan/cat5000/rel_4_2/config/vmps.htm`.