ABSTRACT

ANJUM, IFFAT. Removing Trust within On-Premises Enterprise Networks. (Under the direction of Dr. William Enck).

Traditional enterprise security relies on network perimeters to define and enforce network security policies. This lack of defense-in-depth enables both ransomware and advanced persistent threat (APT) adversaries to advance toward critical targets inside the network. Emerging application-focused Zero Trust architectures attempt to address this long-standing challenge by moving business applications to the cloud and performing enhanced identity and access control checks within a web gateway. However, not all on-premises resources can be relocated to the cloud (e.g., workstations, development servers, file servers, and device management interfaces), and complex forwarding devices remain a high-value target for advanced attacks.

This dissertation proposal seeks to secure on-premises resources that remain as enterprises transition to zero-trust architectures. We consider the two scenarios: 1) compromised hosts and 2) compromised forwarding devices. For the first scenario, we propose Network Views (abbrev. NetViews) for least-privilege network access control where each host has a different, limited view of the other hosts and services within a network. NetViews defines defense at the granularity of individual devices, embracing the communication needs of those devices and extending the existing flexible and dynamic enterprise role- and attribute-based policies into the network. Furthermore, we propose MSNetViews, which extends NetViews for an enterprise network environment with many geographically distributed sites. Each site operates independently and enforces a site-specific *policy slice* that is dynamically parameterized with user location as employees roam between sites. We demonstrate the utility and performance of NetViews and MSNetViews by building prototypes on top of an off-the-shelf reactive Software Defined Network (SDN) platform and emulating realistic environments. Our prototypes have negligible overhead on top of the baseline reactive SDN platform and drastically reduce the attack reachability graph for compromised hosts.

In the second scenario, we consider an attacker with the vantage point of forwarding devices, which can passively identify high-value resources. We propose HoneyRoles, which uses honey connections to deceive and dissuade such adversaries. We build metaphorical haystacks around the network traffic of client hosts belonging to high-value organizational roles. The honey connections also act as network canaries to signal network compromise, thereby dissuading the adversary from acting on information observed in the network. We design a prototype implementation of HoneyRoles using an OpenFlow SDN controller and evaluate its security using the PRISM probabilistic model checker. In doing so, we show that role-based network deception is a promising approach for defending against adversaries in compromised network devices.

Removing Trust within On-Premises Enterprise Networks

by
Iffat Anjum

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Computer Science

Raleigh, North Carolina
2023

APPROVED BY:

Dr. Rudra Dutta	Dr. Bradley Reaves
Dr. Alexandros Kapravelos	Dr. William Enck Chair of Advisory Committee

DEDICATION

To my Parents.

BIOGRAPHY

Iffat Anjum was born and raised in Dhaka, Bangladesh. She is from a Muslim middle-class family. She is the first generation of her family to pursue a career in computer science and obtain higher education. Iffat received her Bachelor of Science in Computer Science and Engineering from the University of Dhaka in June 2013. She also got her Master in Science degree from the University of Dhaka in July 2015. Her MSc thesis was in the area of Cognitive Radio Networks. Over the years, Iffat has accumulated substantial teaching and mentoring experience. She served as lecturer for more than four years at BRAC University and the University of Dhaka. Iffat joined the doctoral program at North Carolina State University in August 2018. Her PhD research leverages emerging technologies, like software-defined networks, to design novel security enhancements for networks and systems. Her work has been published at different security conferences and recognized with the best paper award at the SACMAT. After successfully completing her doctoral degree, she joined as an assistant teaching professor at the University of Denver, Colorado.

ACKNOWLEDGEMENTS

I want to thank my advisor, Dr. William Enck, for his guidance and support throughout my years of graduate school at North Carolina State University. His mentoring, advice, unconditional support, and understanding helped me evolve and advance as an independent academic researcher and was one of the leading factors for the successful completion of my Ph.D.

I thank Dr. Bradley Reaves for his guidance, advice, and expertise. His perspectives and insights on approaching and conducting research and career have had a significant positive impact on me. I want to thank my collaborators, specifically Dr. Cristina Nita-Rotaru, Dr. Mike Reiter, Dr. Munindar Singh, Dr. Christopher Kiekintveld, Daniel Kostecki, Jessica S., Ramzah Rehman, Dr. Isaac Polinsky, Mu Zhu, Dr. Mohammad (Sujan) Miah, for their efforts and guidance. Next, I would also like to thank my doctoral committee members, Dr. Alexandros Kapravelos, and Dr. Rudra Dutta, along with the members mentioned above, for their feedback and discussions during the course of my degree.

Over these past five years at North Carolina State University, I had the chance to interact with many exceptional individuals in the Wolfpack Security and Privacy Research (WSPR) Lab. I thank all the faculty members for their time and support throughout the years, specifically Dr. Alessandra Scafuro, Dr. Anupam Das, and Dr. Laurie Williams. I would also like to thank all of my labmates, both past and present, for their support and expertise, specifically Sathvik Prasad, Nusrat Zahan, Trevor Dunlap, Virgil English, Samin Yaseer Mahmud, Nikolaos Pantelaios, Shaown Sarker, Ramzah Rehman, Dr. Isaac Polinsky, Abhinaya Srividhya Balaji, Seaver Thorn, Dr. Igibek Koishybayev, Dr. Abida Haque, Dr. Benjamin Andow, Dr. TJ OConnor, Dr. Adwait Nadkarni, and Sanket Goutam.

I am grateful to Dr. Md. Abdur Razzaque for his guidance and for introducing me to research during my undergraduate and master's degrees at the University of Dhaka. I likely would not have considered pursuing a Ph.D. had it not been for his guidance.

I can not be thankful enough to my spouse, Mohammad Maruful Haque, and my mother, who always believed in me and kept my hopes up even in the darkest days. I would also like to thank my daughters, Manha and Mahira. You both have been my source of light and happiness. I love you both dearly. Finally, I am grateful to my close friends and family for their unwavering support throughout these long academic years.

The works in this dissertation were partly supported by ONR grant N00014-20-1-26969 and ARO grant W911NF-17-1- 0370. Any findings and opinions expressed in this material are those of the author and do not necessarily reflect the views of the funding agencies.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

Traditionally, network security has followed the "moat-and-gate" defense, where accessing the network resources from outside is hard, but everything inside is trusted by default. The fundamental drawback of this approach is the assumption that an attack can only happen from outside. Hence, if an attacker gains access to the inside network, it can access all network resources. The vulnerability of this security approach intensifies by the fact that enterprises no longer have their resources and user in one physical location. The network architecture has become so complex that perimeters are no longer a practical definition of security.

After decades of criticisms for "moat-and-gate" defenses, enterprise network security is on the verge of a fundamental change. Practitioner interest in Zero Trust [123] has reached a tipping point, recently motivated by requirements stated in US Whitehouse Executive Order EO-14028 [69] and Memo M-22-09 [119]. Zero Trust models assume the attacker has already breached perimeter defenses. Hence you can not trust anything without verifying. The predominate Zero Trust models (e.g., BeyondCorp [148] and BastionZero [158]) require organizations to place critical business applications on cloud servers where web application gateways perform multi-factor authentication, device attestation, and behavioral analytics. However, not all on-premises resources can be relocated to the cloud, and those remain a high-value target for advanced attacks such as Solorigate [49] and NotPetya [13].

Enterprises heavily rely on the security of their on-premises network resources, including desktops, laptops, servers, routers, and switches. By compromising one or more of these resources, an adversary may cause significant harm to the enterprise. Attackers can perform active and passive reconnaissance to identify high-value targets. If done strategically and sparingly, such manipulation can fall under the detection thresholds of existing defenses. Furthermore, in an enterprise environment a range of activities are performed through on-premises resources by different types of users with different roles (e.g., IT administrators, C-suite executives, and finance personnel) [26]. The roles of the user (and hence the host) can change over time as the user takes on new responsibilities and transitions away from old ones. Attempting to bin

these roles as security parameters (e.g., perimeter definition) often results in a more permissive security policy than is needed, thus decreasing security.

Including enterprises with geographically distributed sites in the scenario leaves the existing perimeter-based defenses more wanting. With this multi-site scenario, globally managing user context and access policy across an organization is essential, yet intractable and error-prone. For example, Memo M-22-09 [119] states that "Zero trust architectures require metadata about the user to allow agencies to make risk-based decisions at the policy enforcement point. That metadata is maintained, updated, and supplied by systems that manage user identities, keeping the appropriate metadata associated with the correct user *even if that user leaves the organization or moves to a new position within it.* ... Using *centrally managed* systems to provide enterprise identity and *access management* services reduces the burden on agency staff to manage individual accounts and credentials." (emphasis added). Thus, the goal of enforcing zero-trust in the on-premises network is essential for maintaining a secure environment. However, this goal is a long way from being achieved.

1.1 Thesis Statement

This dissertation focuses on enforcing the idea of zero-trust in the on-premises enterprise network environment. While working towards this broader goal, we identify the potential of reactive Software Defined Network (SDN) towards addressing the threats to on-premise networks. Part of the original vision of SDN was to handle access control decisions based on the flow from or to an individual host. This can reduce the network perimeter to the level of individual hosts. Reactive forwarding can overcome the limitation of untrusted access networks by providing protection to all legacy devices and maintaining compatibility with legacy services. However, more sophisticated forwarding devices can themselves become an attack vector. An attacker can perform *passive* reconnaissance by compromising packet forwarding devices and inspecting network flows to identify the existence and behaviors of client and server hosts. In this dissertation, we utilize reactive forwarding to design and enforce the least privilege access control and deception. We hypothesize a combination of these security mechanisms enable the creation of zero trust security for on-premises enterprise network and address the problems created by the changing network architecture. This leads us to the following thesis statement:

Role-based access-control and deception via reactive forwarding on 5-tuple network flows is essential for zero-trust within on-premises enterprise networks.

In particular, this dissertation highlights the need to consider user functionality or role in the enterprise in defining network security. It also highlights the importance of considering

the 5-tuple network flow information (source IP, source port, destination IP, destination port, TCP/UDP protocol) in access control decisions and deception techniques. Leveraging the role and 5-tuple information, we first design a defense architecture that addresses the issues of compromised hosts inside a single-site enterprise network. Consequently, we identify the distinct challenges of a geographically distributed multi-site enterprise network and propose a unique system to address these challenges to achieve zero trust. Finally, we consider the insider attacker from the vantage point of a compromised packet forwarding device. We demonstrate that a novel role-based deceptive defense leveraging reactive forwarding can defend against reconnaissance and active attack. All these proposed systems demonstrate the usability and effectiveness of reactive forwarding in defending against insider threats and establishing the ideology of zero trust in on-premises enterprise networks.

1.2 Contributions

In this dissertation we make the following contributions:

- We design and implement a novel paradigm for enterprise network security called Network Views (abbrev. NetViews), where each host has a different "view" of what other hosts and services exist in the network. This fine-grained least-privilege approach to network access control can significantly reduce lateral movement by attackers, even if user credentials have been compromised. NetViews builds on NIST's Next Generation Access Control to provide a dynamic and scalable policy model that supports the needs of large enterprises. We present an SDN-based design and demonstrate that our implementation has network latency and throughput comparable to baseline reactive forwarding. We further provide an optimization for multi-connection flows that significantly reduces both redundant access control checks and forwarding state storage in switches. As such, NetViews offers a practical primitive for removing the reliance on security perimeters within enterprise networks. This work was published at SACMAT 2022. For more information about the NetViews system architecture, see Chapter 3.

- We design and implement Multisite NetViews (abbrev. MSNetViews), which models a system that extends a single-site enterprise network security policy to many geographically distributed sites. Each site operates independently and enforces a site-specific policy slice dynamically parameterized with user location as employees roam between sites. We build a prototype of MSNetViews and show that for an enterprise with globally distributed sites, the average time for the policy state to settle after a user roams to a new site is well below two seconds. As such, we demonstrate that multisite organizations can efficiently protect their on-premises network-attached devices via a single global perspective. This work was published at SACMAT 2023. For more information about the MSNetViews

system architecture, see Chapter 4.

- We design and implement HoneyRoles which uses honey connections to deceive adversaries and dissuade them from performing attacks. The honey connections act as network canaries to bait adversaries and detect their presence more quickly. A key idea behind the HoneyRoles is to focus on client hosts performing high-value organizational roles, building metaphorical haystacks around their network traffic. We built a prototype of HoneyRoles in an SDN environment and modeled its operation using the PRISM probabilistic model checker. Our performance evaluation shows that HoneyRoles has a small effect on network request completion time, and security analysis demonstrates that once an alert is raised, HoneyRoles can quickly identify the compromised switch with high probability. In doing so, we show that role-based network deception is a promising approach for defending against adversaries in compromised network devices. This work was published at CODASPY 2021. For more information about the HoneyRoles system architecture, see Chapter 5.

1.3 Thesis Organization

The remainder of this dissertation proposal is organized as follows. Chapter 2 presents the background and literature details needed to comprehend the proposed works. Chapter 3 presents NetViews, and Chapter 4 presents MSNetViews, proposing a granular least privilege access control and enforcement to invoke perimeter-less access control for single-site and multi-site enterprise networks. Finally, Chapter 5 describes how we deploy honey connections to deceive and dissuade the attackers with the vantage point of forwarding devices.

Chapter 2

Background and Related Work

This chapter describes the foundations and state-of-the-art works of software defined networks, zero trust architecture, access control frameworks, next generation access control, reconnaissance, and deception framework.

2.1 Software Defined Networking

Software Defined Networks (SDNs) have the potential to address many operational and security challenges in enterprise networks [91, 97]. They decouple network control from the underlying data plane and consolidates configuration to a logically central controller, which provides valuable flexibility for dynamic traffic forwarding [101]. In contrast to proactive SDN (statically installing rules in switches), reactive SDN provides a flexible reference monitor interface [7] for limiting attackers' movement once they gain access to a host within an enterprise network. Initially, SDN switches have no forwarding state. When a switch receives a packet that does not match any forwarding rules, it sends a `PacketIn` message to the SDN controller, as shown in Figure 2.1. The logically central controller uses multiple applications (e.g., forwarding, access control) to determine to which physical port the switch should forward the packet. Its response to the switch usually takes the form of a `FlowMod` message, which defines a forwarding rule matching future packets. Along with each rule comes a predefined expiry time at which the switch removes the rule from its forwarding table.

Numerous controller designs and implementations have appeared over the years [90, 57, 50]. Among them, the leading controller is the Open Network Operating System (ONOS) [56], developed and maintained by the Open Networking Foundation. The latency bottleneck in SDN operations comes from `PacketIn` messages, which are not dataplane operations, and thus do not move the data packets any closer to their destination. Researchers and engineers have designed several optimizations to address this problem. One optimization that is relevant for our work is that, in some cases, it is possible to determine the entire forwarding path from the first

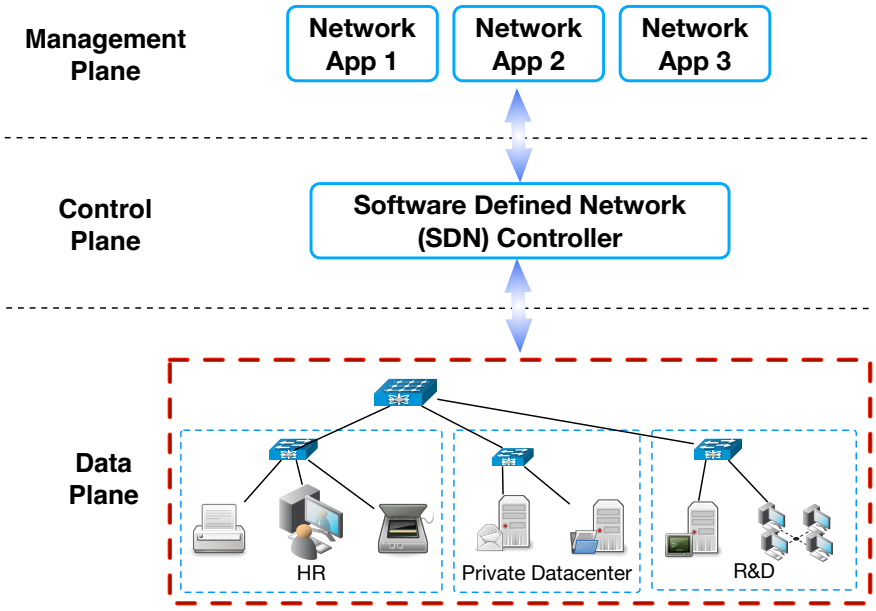


Figure 2.1: Basic structure of SDN

`PacketIn` and proactively send `FlowMod` messages to the subsequent switches. In fact, many SDN controllers provide this *one-big-switch* forwarding primitive. In ONOS, it is called an *Intent*. The advantage of this technique is that the controller does not have to wait to process `PacketIn` events from the subsequent switches on the forwarding path—it already knows everything it needs to make forwarding decisions for the entire path.

SDN has the potential to supplant conventional security systems [154], simplify policy enforcement [122, 62], ensure information flow control [117], enable deceptive defense [107], and provide a software defined perimeter [36, 112]. However, the greater capabilities and open functionality of SDN switches increase the potential for compromise and enable a new vantage point for attacks, e.g., data plane attacks using advanced reconnaissance, data manipulation, and redirection (e.g., Teleportation [141], Benton et al. [18], Menghao et al. [159], Know Your Enemy [37]).

Sphinx [39] uses SDN control messages for incremental validation of network updates and detect suspicious behaviors (e.g., DoS, blackholing, fake topology). WedgeTail [131] detects both forwarding attacks and forged packets by utilizing Header Space Analysis and other network troubleshooting tools. Both Sphinx and WedgeTail dynamically construct network flow graphs to compare with a defined policy to identify deviations, which is not only a manual and error-prone process but also cannot handle dynamic networks. Additionally, DynaPFV [93] proposed a mechanism to detect packet-modification by comparing the cryptographic hash of packets at the ingress and egress points of a network. However, none of these prior works can address passive (or even subtle active) reconnaissance. Since reconnaissance can be performed without

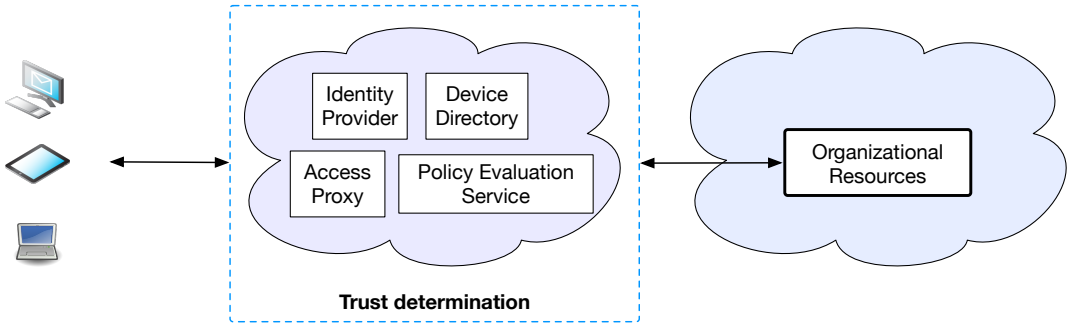


Figure 2.2: Cloud based ZTA solutions

network disruption attacks (e.g., forwarding, packet forging, and packet-modification attacks detected by the tools above), the attacker is able to evade the defenses of prior works.

Network analysis and auditing tools (e.g., Header Space Analysis [79], VeriFlow [81], SDN-RDCD [160]) can protect against network or SDN controller configuration failures (or attacks). However, a compromised SDN data plane can introduce different types of attack scenarios [11], which are not possible to detect through header flow analysis alone. Some solutions have sought to detect forwarding attacks by monitoring flow statistics from neighboring switches [114], verifying OpenFlow events in the controller [159, 147], applying heavy-weight cryptographic approaches [84], and naive controller generated probes [31].

2.2 Zero Trust Architecture (ZTA)

The COVID-19 pandemic has driven an enormous paradigm shift to the old castle and moat model. These days the castle is often empty, with many workers connecting to resources remotely [73]. ZTA is becoming the gold standard of enterprise security, where no trust between any entities is assumed unless explicitly specified [124]. ZTA enforces the principle of least privilege for securing "protect surface", enterprise data, assets, applications, and services [85]. The goal of Zero Trust is to "*prevent unauthorized access to data and services* coupled with making the *access control enforcement as granular as possible*" [123]. With zero trust you can not allow a device to talk with other just because they are in the trusted perimeter. Every user and device must be verified, and only after that they should have access to a small set of resources that necessary to complete their job. However, enabling zero trust is not straightforward for any enterprise networks (e.g., web-based solutions fail to address legacy systems).

Government authorities (such as US White-house memo M-22-09 [119], NIST SP 800-207 [123]) have, in recent days, emphasized the importance of ZTA. Google's BeyondCorp [148] has led the industry conversation on Zero Trust, which has inspired subsequent efforts including Software Defined Perimeters (SDP) by the Cloud Security Alliance [109] and BastionZero [158]. Academic literature also leveraging ZTA concepts for secure network environments (e.g., server-

to-server access in PagerDuty [42], risk-based access control [145], and access control for 5G networks [19]).

Most ZTA efforts concentrate on web applications and provide cloud-based solutions (Figure 2.2). These models require organizations to place critical business applications on cloud servers where web application gateways perform multi-factor authentication, device attestation, and behavioral analytics. Although there is great value in moving business applications from servers on enterprise-premise to cloud services, more is needed to secure on-premises devices and services. Not all on-premises resources can be relocated to the cloud (e.g., development servers, file servers, and device management interfaces), and employee workstations remain a high-value target for advanced attacks such as Solorigate [49] and NotPetya [13]. Hence, how ZTA can be incorporated in securing the on-premises entities remains an open question.

2.3 Access Control Frameworks

As computer systems contain a large amount of information, it's necessary to define what entities have access to what information through well-defined access control frameworks. Access control systems most commonly seek to achieve *least privilege*, where every program and user of the system should operate using the least set of privileges necessary to complete a corresponding task [129]. Conceptually, least privilege policies can be expressed as an access control matrix; however, access control lists (ACLs) commonly provide more efficient policy storage. To simplify network management challenges, many systems and proposals (e.g., Ethane [27], FML [65]) have used group-based policy definition, which is mostly static and fails to represent enterprise configurations [127, 126]. Role-Based Access Control (RBAC) [126] assigns permissions to roles to simplify policy management and models users' functional rules within an organization. However, traditional RBAC is less suitable when multiple features of users and devices are required for access control decisions [38]. To address this limitation, attribute-based access control (ABAC) [70] provides logical, fine-grained, and context-aware access control. Several extensions of ABAC, including NGAC [48] and XACML [47], have been proposed and used mainly to manage file-based resources and web services. We are unaware of existing commercial or research access control solutions that provide a unified and granular solution for modern-day network security [113, 148].

2.4 Network Access Control

Historically, both the external and the *internal* perimeter of an enterprise network was secured by firewalls [30, 61]. Devices within a certain protected perimeter are usually trusted and leverage complete connectivity. However, firewalls are hard to configure and maintain, particularly in scenarios with multiple domains or layers to secure [100, 151]. Furthermore, commercial next-

generation firewalls are just firewalls with NIPS (Network Intrusion Prevention System) [115]. Modern enterprise network operation, access control, and security are currently maintained through different ad-hoc mechanisms: by building Layer-2 or Layer-3 boundaries within local networks (e.g., VLAN [136], subnets); through special-purpose devices to control packet flow (e.g., NIDS [17, 138] or Middleboxes [155, 43]); and hypervisor based systems that combine different state-of-the-art technologies (e.g., SDN) to provide isolation (e.g., FlowVisor [132], PSI [155], micro-segmentation [110, 63, 86]). Furthermore, network segmentation (including micro-segmentation [110, 63, 86]) just puts up more barriers against an attacker.

While prior work [27, 113, 155, 82, 53] provides dynamic policy enforcement (sometimes as a secondary feature), it has limited or nonexistent policy models. Alpaca [77] incorporates roles into IP address assignments to enable efficient packet enforcement in forwarding devices; however the resulting number of roles is limited and it constrains IP address assignment. NEUTRON [143] and SPRT [78] consider least-privilege access control within a network; however, their contributions are focused on creating and testing access control policies and hence are complementary to HoneyRoles. NEUTRON and SPRT also assume traditional network segmentation or micro-segmentation for firewall placement while integrating OpenFlow as future work. Finally, several systems [67, 83] have proposed incorporating the dynamic context from the enterprise network to create context-aware access control. Such context can also be incorporated multiple policy classes supported by NGAC.

2.5 NIST Next Generation Access Control

NGAC is a domain-specific language for writing access control policies. Below we provide a brief introduction to its concepts. For more information we refer the reader to the NGAC article [48].

NGAC policies are defined in terms of sets of objects and actions on those objects. An NGAC policy $\mathcal{P}$ consists of a set of *policy elements* PE, a set of obligations $\mathcal{O}$, and three sets of *policy relations*. The policy relations are: $\mathcal{R}_e$ (assignments), $\mathcal{R}_a$ (associations), and $\mathcal{R}_p$ (prohibitions). The set of policy elements PE includes the sets of users U, user attributes UA, objects O, object attributes OA, and policy classes PC. All objects are object attributes ($O \subseteq OA$).

Policy elements inherit privileges from other policy elements. In NGAC, the set of assignment relations $\mathcal{R}_e$ is a set of pairs of policy elements. It models the graph-based inheritance between policy elements. For $e_1, e_2 \in PE$ and $(e_1, e_2) \in \mathcal{R}_e$, we denote e_1 inherits privileges associated with e_2 by writing $e_1 \rightarrow e_2$. This relation indicates that any privilege assigned to e_2 will also be held by e_1. The assignment relation defines an upside-down tree structure, of which the policy classes PC are roots.

Users are granted access privileges to objects through association relations. The set of association relations $\mathcal{R}_a$ is a set of triplets of user attributes, access rights (privileges), and object attributes. Let R be a set of privileges, if $ua \in UA$ and $oa \in OA$, then we write $(ua, R, oa) \in \mathcal{R}_a$

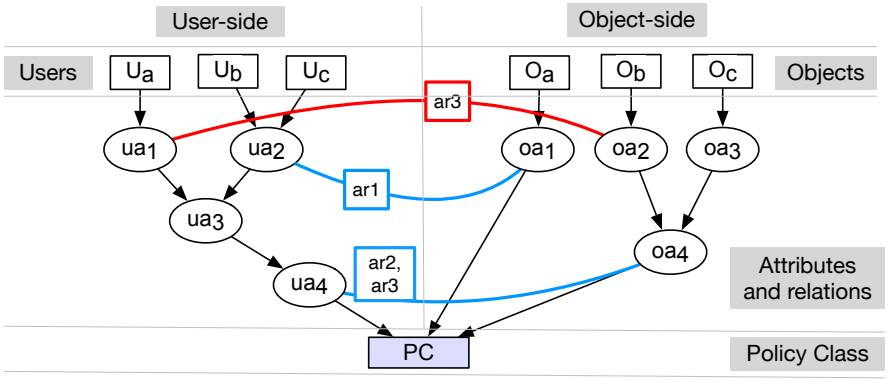


Figure 2.3: Example of NGAC Policy Structure

to indicate that all users $u \in U$ with an assignment path $u \rightsquigarrow ua$ can perform rights R on all objects $o \in O$ with an assignment path $o \rightsquigarrow oa$. NGAC requires association relations *for all* policy classes for an action to be granted. The allowed set of rights is the *intersection* of the maximum sets of rights for each policy class. Formally, for each policy class $pc \in PC$, there must exist an association $(ua, R, oa) \in \mathcal{R}_a$ such that there are paths $u \rightsquigarrow ua$, $ua \rightsquigarrow pc$, $o \rightsquigarrow oa$, and $oa \rightsquigarrow pc$. An simple example policy policy structure is presented in Figure 2.3. Downward arcs represent associations.

Users are explicitly denied access to resources using prohibition relations. The set of prohibition relations $\mathcal{R}_p$ is a set of triples of user attributes, privileges, and object attributes. For prohibition relations, the existence of multiple policy classes does not matter, and prohibitions *always supersede* association relations. The upward arc in Figure 2.3 is a prohibition.

Finally, NGAC allows *dynamic* updates to policy elements and policy relations using *obligations*. The set of obligations $\mathcal{O}$ consists of pairs of event patterns ep and responses r to those patterns denoted $\langle ep, r \rangle$. If an specific event (e.g., change of user location) matches the event pattern, the associated responses are immediately executed, changing the state of the policy. These events may be both internal and external to the policy. Internal events consist of the results of a policy evaluation. External events use a policy application programming interface (API) to trigger obligations.

2.6 Network Reconnaissance

Targeted attacks [41, 92] and threats to enterprise network infrastructure [66, 142] continue to increase. Such attacks often begin with a foothold for reconnaissance. Historically, footholds have been client workstations. However, network packet forwarding devices such as routers and SDN switches are becoming prime targets as they offer a valuable vantage point for reconnaissance and their increased complexity leaves them more prone to compromise. From the vantage

point of a compromised network switch, the adversary can perform various en route network traffic attacks that strategically and selectively target high-value clients at critical times. For example, it could inject malicious JavaScript into Web pages as they are returned from Web servers, or it could use SSL-stripping to eavesdrop on traffic and steal credentials.

Traditional intrusion detection systems cannot detect passive attackers performing network reconnaissance from compromised packet forwarding devices. If done strategically and sparingly, these can easily fall under the detection thresholds of existing defenses [39, 31, 131]. Such reconnaissance investments are particularly apropos to advanced persistent threats (APTs) [29]. Especially, Advanced Persistent Threat (APT) owns a long cyber kill chain, which includes reconnaissance, delivery, initial intrusion, lateral movement, and data Exfiltration [29]. After gaining a foothold in a network by compromising a device, adversaries can leverage information through reconnaissance to better identify targets and vulnerable devices. Bartlett et al. [15] demonstrate the dangers of reconnaissance by presenting a quantitative comparison and evaluation of the effectiveness of passive monitoring and active probing for service discovery in decentralized networks.

Even if traffic is encrypted, reconnaissance remains a threat. Schuster et al. [130], Backes et al. [14], and Ling et al. [94] show that encrypted web traffic can leak information through packet length, packet timing, web flow size, and response delay. Similarly, AppPrint [105] analyzes the possibility of fingerprinting mobile apps via comprehensive traffic observations. Anderson et al. [6] have produced TLS fingerprint from network data, revealing the details of TLS versions and other configuration parameters. Schuster et al [130] demonstrates the technique of identifying encrypted video streams using the latency between the packet streams. With increasing threats of targeted reconnaissance and attacks (e.g., Snowden [134], CISCO SYNfulKnock [66], political espionage [92]), defense against APT is becoming more critical.

Existing defenses such as HSTS have seen limited deployment [87], in part because many developers do not understand how to use HSTS correctly, resulting in critical information such as login cookies being leaked. For networks that include mobile devices, Luo et al. [98] found that popular mobile web browsers failed to fully support HSTS and were left open to clickjacking attacks. Additionally, Krombholz et al. [88] showed that TLS deployment is far too complex, leading to large numbers of incorrect HTTPS deployments.

2.7 Deceptive Defenses

Deception techniques provide alternative defense approach that can mislead and delay adversarial efforts, and even detect attacks in early stages. Whaley et al. [149] define deception as the misperception that is intentionally induced by other entities. Spafford et al. [4] define cyber-deception as "planned actions taken to mislead and/or confuse attackers and to thereby cause them to take (or not take) specific actions that aid computer-security defenses." Current decep-

tive defense techniques primarily use a moving target defense (MTD) approach. These solutions depend on mimicking random or static specification of system behavior, network configuration, or network infrastructures (e.g., honeypots, honey-nets) [68, 12, 156, 72, 125, 157]. On the other hand, dynamically generated decoy traffic [21], decoy based IP randomization [34], and decoy vulnerability-based honey traffic [10] provide more effective defense.

The dynamic control and programmability of an SDN environment has inspired new deception techniques. HoneyMix [64] uses a dynamic SDN-based honey-net to automate interactions with adversaries, and showed deception is a promising approach toward defending against network reconnaissance. Further, the dynamic network configuration of an SDN can be used for discriminating against scanning attacks and enhancing targeted defenses [99, 10]. For example, Achleitner et al. [1] use SDN to defend against insider reconnaissance by simulating virtual network topologies as decoys. SDN also enables the dynamic network configuration for discriminating against scanning attacks and enhancing targeted defenses [99].

Chapter 3

Network Views

In this chapter, we present *Network Views* (NetViews for short) as an abstraction and model for access control within enterprise networks that provides a fine-grained least-privilege network access control. NetViews redefines network security for on-premises enterprises and government networks to focus on individual devices; This is the building block for achieving zero trust for on-premises enterprises. In this chapter, we are focusing on single-site enterprise networks, and Chapter 4 will extend it to handle the scenario of multisite enterprise networks.

3.1 Introduction

Enterprise networks traditionally rely on perimeters for defense. Perimeters provide boundaries both at the WAN access edge as well as at critical points within the network. However, this moat-and-gate approach for network access control has not aged well as the functional needs of enterprise networks have evolved. This lack of defense-in-depth enables both ransomware and advanced persistent threat (APT) adversaries to advance towards critical targets inside the network (e.g., Solorigate [49], NotPetya [13]).

Zero Trust architectures (e.g., NIST SP 800-207 [123]) remove the reliance on network perimeters for defense. The Zero Trust model promoted by Google's BeyondCorp [148] and the recent US Whitehouse memo M-22-09 [119] focuses on applications, removing the network from consideration altogether. In this idealized environment, a cloud-based web application gateway authenticates users, the accessing device, and uses behavior analytics to determine if the request should be forwarded to the target web application.

However, even if business applications can be moved into the cloud, on-premeses networks cannot be ignored. Enterprise networks often contain on-premises development servers, file servers, and device management interfaces. Furthermore, both Solorigate and NotPetya spread using network services commonly left open on workstations, motivating stronger defenses *within* networks. Indeed, both SP 800-207 and M-22-09 note the potential need for logical micro-

segmentation as an isolation strategy. However, micro-segmentation does not solve the root of the problem. Rather, it exposes the complex communications needs of real networks, requiring network engineers to continually redefine new network boundaries and firewall rules between them. In contrast, we argue that *networks should define defense at the granularity of individual devices,* embracing the communication needs of those devices and extending the existing flexible and dynamic enterprise role- and attribute-based policies into the network.

The technology needed to achieve this vision exists and is commercially available. Software Defined Networking (SDN) technologies such as OpenFlow [101] and P4 [20] provide opportunities to create flexible and reactive policies. However, prior work has focused primarily on enforcement mechanisms, identifying novel ways of using forwarding rules to improve performance (e.g., FlowTags [43], PSI [155], Alpaca [77], Kinetic [82]). Proposals that do consider access control (e.g., Ethane [27], FML [65], Alpaca [77]) do not capture the dynamicity and scalability requirements of enterprise environments.

In this chapter, we propose *Network Views* (NetViews for short), where each host has a different "view" of what other hosts and services exist in the network. We designed and built a prototype of NetViews and find that even on a heavily loaded network, both new and established flows have latency and throughput comparable to baseline reactive forwarding. Our security analysis on a concrete reference topology demonstrates the effectiveness of NetViews in reducing reachability, improving overall security.

We make the following contributions in this work.

- *We propose an access control model supporting the NetViews abstraction.* The model is based on NIST's Next Generation Access Control (NGAC) policy language [48] and allows flexible management of network connectivity.

- *We design and implement NetViews as an SDN application.* NetViews leverages ONOS's Intent [55] framework for efficient management of forwarding rules in switches. Our Mininet-based performance evaluation using three representative topologies shows latency and throughput comparable to baseline reactive forwarding.

- *We provide a multi-connection optimization that significantly reduces redundant access control policy checks and forwarding state in switches.* We show that a 5-tuple flow definition (s_{ip}, s_{port}, d_{ip}, d_{port}, $proto$) of access control is unnecessarily strict, and that allowing any s_{port} does not sacrifice security. This optimization significantly reduces the tertiary content addressable memory (TCAM) requirements for switches.

We note that the NetViews is designed for *intra*-network access control and is not a replacement for firewalls at the network edge. While NetViews currently only uses NGAC policy when making access control decisions, its policy engine can be easily extended to incorporate

additional logic. For example, with future developments and strong threat-analytics NetViews can be a primitive that enables ZeroTrust for any type of networks.

Note that throughout the chapter we assume the enterprise network has been fully provisioned with a reactive SDN technology such as OpenFlow. However, a full deployment is not necessary in practice. Brockelsby and Dutta [23] recently found that most network traffic in access switches in university networks is north-south. In doing so, they show that legacy switches with VLAN capabilities can be used to isolate all traffic in the access switch layer, forcing it to a more capable distribution switch with SDN capabilities. We leave the exploration of such architectures to future work.

Availability: The source code for our NetViews implementation is available at `https://github.com/netviews/ss-netviews`.

3.2 Motivation

The goal of Zero Trust is to "*prevent unauthorized access to data and services* coupled with making the *access control enforcement as granular as possible*" [123]. Emerging application-focused solutions such as BeyondCorp [148] move business applications to the cloud and place them behind web gateways that perform enhanced identity and access control checks. However, such application-focused solutions ignore the workstations, development servers, and device management interfaces that remain in the on-premises network. Figure 3.1 shows an example enterprise network consisting of several floors of a building. Consider a traditional network architecture where the dashed boxes represent coarse security domains, where firewalls enforce access control at the perimeter of each domain.

Organizations struggle with the growing threats of ransomware and APT attacks, where the attacker initially gains access to a single host. The attacker then is able to move freely within the perimeter until it finds a host that is allowed to connect to a host in a different perimeter. It progressively moves across the network until it achieves its goal of accessing high-value information and resources like customer data or code repositories.

NotPetya, a malware attack that dominated the year 2017 [13, 33], took advantage of the lack of least privilege policy and granular enforcement. NotPetya propagated through a combination of vulnerabilities in the Microsoft SMB service running on specific TCP ports (tcp/139, tcp/445) as well as compromised user credentials. These stolen credentials gave the attacker additional authority to exploit other reachable hosts via the vulnerable file and network information sharing service. Later, in the 2020 Solorigate event [49, 103], attackers used a remote PowerShell (tcp/5985, tcp/5986), which is often enabled on the hosts to allow administrators to scale administration tasks. However, due to of the lack of the least privilege policy within the network, those remote PowerShell commands could originate from anywhere in the network, not necessarily from machines associated with specific network administrators. These exam-

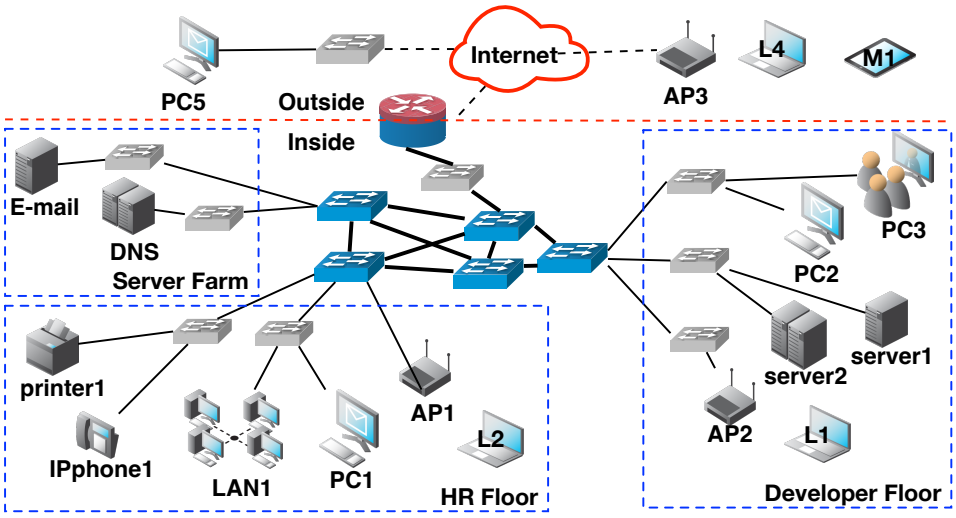


Figure 3.1: Example Enterprise Network

ples demonstrate the need for a new network access control abstraction to severely limit or slow down lateral movement. In both cases, the specific network ports on hosts were left open for administrative purposes. While host-based firewalls could theoretically limit access by IP address, managing those in mass deployments is complicated and error-prone. In contrast, a network-centric enforcement of fine-grained, least-privilege policies would significantly reduce the attack surface.

Enforcing fine-grained, least-privilege network connections between hosts requires a corresponding access control policy. Existing methods of specifying firewalls will not scale to the tens of thousands of potential connections between clients and server ports. Furthermore, the policy will constantly change as personnel roles and organizational objectives are updated over time. Consider a developer, Alice, who sits on the developer floor (Figure 3.1) working from `L2` and needs access to Git on `server2`. Right before a meeting with her manager, who is on the HR floor, Alice needs to print a project report on `printer1`. Alice's meeting went exceptionally well, and she is promoted to a management position with a large office on the HR floor. However, during the transition, Alice still needs access to the Git server on `server2` to finish a project, as well as to be available to fix bugs until she can train a replacement. There can be multiple users like Alice in the enterprise, which makes the overall security maintenance complex, and can lead to inconsistencies among different perimeters.

Threat Model and Assumptions NetViews seeks to provide an access control framework for enterprise environments (e.g., government, university, military, corporate). We assume both insider (e.g., authenticated users, services) and external (e.g., exploited software, IoT devices) attackers. The goal of attackers is to compromise devices, exfiltrate data, or disrupt enterprise services. Attackers may compromise user passwords and multiple hosts, pivoting through the

network to accomplish their goals. Our trusted computing base includes all implementation components of NetViews. We assume the SDN infrastructure, including controller and switches, is securely deployed (e.g., out-of-band or TLS-protected southbound communication) and free of errors. We assume all installed SDN applications are benign and free of errors. We also assume appropriate defenses are deployed to mitigate known DoS concerns for reactive forwarding (e.g., `PacketIn` flooding). Finally, we assume the enterprise network has a secure and robust authentication mechanism for users and services.

3.3 Overview

We seek to restrict the lateral movement used by APTs and ransomware by enhancing network environments with granular, least-privilege access control of network flows between hosts. Achieving this goal requires overcoming the following research challenges.

- *Policy must apply to every network packet.* Achieving complete mediation of network traffic requires the ability to inspect every packet sent by and received from every host. Forcing traffic through choke-points induces unnecessary latency and reduces network reliability.
- *Policy enforcement must be fine-grained.* At fine-granularity, not all of the policy can be precomputed (e.g., client source ports). Even if it could be precomputed, the policy size would be too large to store in network switches.
- *There is a semantic gap between networking primitives and an enterprise's organizational structure.* The roles and duties of employees change in response to organizational needs. Network administrators should not be expected to manually translate changes in roles into changes in firewall policy.

Reactive SDN architectures such as OpenFlow provide the ability to perform complete mediation of all packets sent by and received from every host without forcing traffic choke-points. In a reactive setup, switches send `PacketIn` messages to a logically central controller whenever they encounter a packet that does not match an existing forwarding rule. The controller responds to a `PacketIn` message with a `FlowMod` message, which tells the switch which port to forward the packet to as well as a rule for matching future packets. Thus, reactive SDN only incurs additional latency on the first packet for the rule, and subsequent packets are forwarded at line-speed. The need for additional `PacketIn` messages depends on the granularity of the match rule, which can be coarse (e.g., based on a CIDR prefix or MAC address) or fine-grained (e.g., based on a 5-tuple of source IP, source port, destination IP, destination port, transport-layer protocol). Switches have a limited amount of TCAM for storing flow rules, and old rules may be

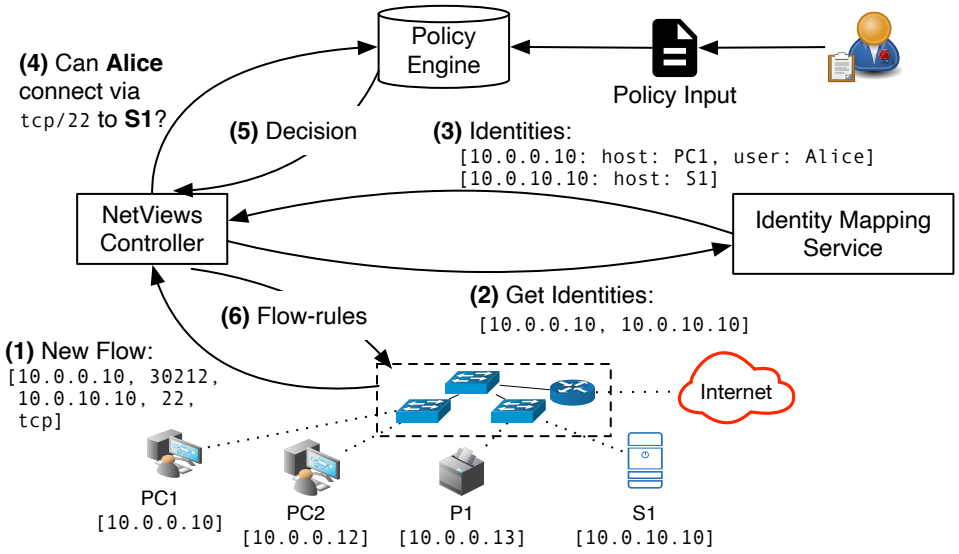


Figure 3.2: Overview of NetViews design portraying the necessary components and a possible interaction among them.

expunged if the TCAM fills. Therefore, care must be taken when using fine-grained forwarding rules.

Prior work [27, 113, 65] has proposed reactive SDN for enforcing access control of network flows between hosts. However, their policy models are either non-existent or based on groups, which cannot reflect the permission hierarchy and dynamic nature of an enterprise [127, 126]. Alpaca [77] incorporates roles into IP address assignments to enable efficient packet enforcement in forwarding devices; however, the number and granularity of roles are limited. In contrast, we adopt NIST's NGAC model [48, 46], which can describe role (RBAC) and attribute (ABAC) based policy models, as well as a range of dynamic policy features. We overcome the performance limitations of prior fine-grained match rules by (1) building on top of the Intent primitive in the ONOS SDN controller to proactively send `FlowMod` messages to all switches on a forwarding path as soon as the `PacketIn` message at the first switch is received; and (2) using a multi-connection optimization that slightly expands the match rule without sacrificing security. In doing so, we envision an enterprise can simply extend their existing organization policy into the network.

Figure 3.2 shows a high-level overview of the NetViews design depicting the key logical components: (1) the NetViews controller, (2) the identity mapping service, (3) the policy engine, and (4) an SDN data plane with reactive forwarding. When the data plane encounters an unknown flow, the NetViews controller receives the event (Step 1). The NetViews controller then queries the identity mapping service to translate flow IP addresses into the user and host information referenced by the policy (Steps 2 and 3). Next, the NetViews controller queries the policy engine using the derived user and host, as well as the Layer-4 connection information, e.g., `tcp/22`, (Step 4). If the policy decision is *deny*, then the flow is dropped. However, if the policy decision is *allow* (Step 5), the NetViews controller installs forwarding rules in all switches

on the determined path from the source to the destination (Step 6). Forwarding rules are defined for the network flows in both directions.

3.4 NetViews Policy Model

A key contribution of this chapter is the integration of a flexible access control policy model into a least-privilege network environment. By building upon NIST's NGAC, NetViews benefits from decades of access control research. NGAC's flexibility also allows NetViews to integrate with enterprises currently using both role and attribute based policies, assuming suitable tools to transform RBAC and ABAC policies into an NGAC policy.

Existing policy models such as NGAC, RBAC, and ABAC assume a traditional operating system environment. This section addresses two key questions: (1) *How should NGAC policy concepts capture network primitives while bridging the semantics?* And (2) *What are the semantics of an* allow *decision and also how should the networking infrastructure respond to an allow decision?*

NGAC was designed to protect resources in a traditional OS context where users, objects, and access rights are obvious and the reference monitor need only mediate the initial request. In designing NetViews, we explored a range of options. The remainder of this section describes several potential alternatives and explains the rational for our end design. Note that our contribution is not changes to the NGAC policy model itself, but rather in how to apply NGAC to network primitives.

3.4.1 Users, Objects, and Access Rights

The NGAC model [48] defines authorized users (U), processes (P), objects (O), user and object attributes (UA and OA), policy classes (PC), operations (Op), and access rights (AR). A key question for our work is how to encode these essential elements with network access control concepts. An brief overview NGAC policy model can be found in Section 2.5.

Users: We primarily focus on user workstations. We assume that each host on the network has at most one user at any given time, and a single user entity can access the network from different hosts. We consider physical and virtual hosts in an enterprise network setting, each associated with IP and MAC addresses. Traditionally, a network flow is a 5-tuple: (source IP, source port, destination IP, destination port, protocol— e.g., TCP). The source and destination ports are bound to the operating system (OS) process executing on the hosts; however, this process-level context is generally unavailable to network infrastructure. That said, OS processes are associated with users, and in many settings, each host can be associated with a user. For simplicity, we assume that each host on the network has at most one user at any given time. This assumption is realistic, even for shared workstations that require users to login (e.g., via Active Directory), as the domain controller can inform the network infrastructure which user

is associated with a host. For servers, we assume the use of per-user virtual machines, in some cases, the "user" for the server may not be a physical person but rather an abstract user to perform a task. Containers could also be handled similarly, assuming proper bookkeeping by the orchestration platform and perspective into the networking primitives (e.g., IP addresses) within the container platform.

NetViews decides whether or not an *IP packet* should be forwarded. This decision must be based on the network flow 5-tuple information available in the packet and possibly source and destination MAC addresses. We considered several approaches for mapping network environment information to NGAC. For example, should NGAC users map to MAC addresses, IP addresses, hosts, or real users? Given the expressiveness of NGAC, the assignment of users and user attributes to other user attributes can capture the relationships between IP addresses, hosts, and real users. Therefore, we mapped NGAC's definition of *user* with the real $\langle user, device \rangle$ pair and used a separate identity mapping service to translate the network flow 5-tuple to the $\langle user, device \rangle$ pair. However, changing the device does not change a user's role; it can only introduce some extra capabilities or restrictions. For this reason, we decided to maintain a unique user identifier for each user and consider it as a first-hop user attribute. For example, consider a user Alice with two devices ($\langle Alice, L1 \rangle$ and $\langle Alice, PC1 \rangle$) has a unique identity $u1$. In our policy graph, both $\langle Alice, L1 \rangle$ and $\langle Alice, PC1 \rangle$ will have a *assignment* relation with user attribute $u1$.

Objects and Access Rights: Given the identity mapping service, NGAC objects could be hosts or TCP ports on a host. TCP ports map to server applications, which could be suitable objects. Hosts could then be object attributes that contain the TCP port objects. However, this design is incompatible with aspects of network operation. For example, it does not naturally capture how to control access to ARP, which is a prerequisite for most IPv4 flows. Since our goal is to define a model for *visibility*, it is more natural to map NGAC objects to server hosts and manage TCP/UDP ports, ICMP, and ARP as NGAC access rights.

Example NetViews Policy Scenario: Figure 3.3 depicts a small example NetViews policy for the network topology in Figure 3.1. The policy includes a single policy class, a set of users U, and a set of objects O. The policy also shows a set of user attributes UA = {u1, u2, u3, HR-U, DEV-U, local} and a set of object attributes $OA = O \cup$ {domain, mail, printer, server, remote-cpu, SF, HR, DEV}. The black lines with arrows depict *assignments* which correspond to the notion of *containment*. For the object side of the policy, attribute containment indicates that access rights are given to a set of objects.

The figure depicts *associations* as downward blue arcs labeled with a set of access rights. Recall that an association $\langle ua, ars, at \rangle$ specifies that all users contained by user attribute ua have access rights ars for all policy elements contained by at. The figure also depicts *prohibitions* as upward red arcs labeled with a set of access rights. There are several different types of NGAC prohibitions that provide flexibility in how the original set of access rights is restricted.

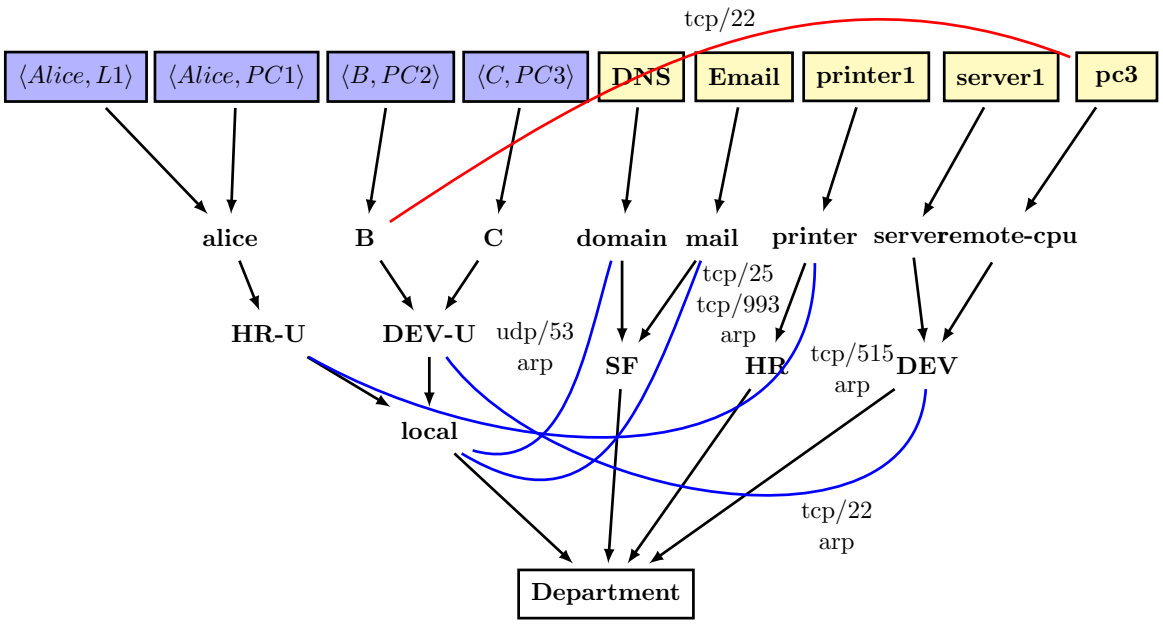


Figure 3.3: NetViews policy example with four user-device pairs (blue) and five resource or objects (yellow). It have a policy class *Department*. Downward blue arcs denote associations and their corresponding rights (e.g., tcp/22). The upward red arc denotes a prohibition and the restricted rights. These relationship arcs determine user privilege over specific objects. For example, $\langle Alice, PC1\rangle$ can access $printer1$ but $\langle Alice, L1\rangle$ can not access it for the prohibition.

Interested readers are referred to the NGAC specification [48] for more details. Note that prohibitions make NGAC's policy model non-monotonic. Semantically, NGAC exhaustively searches all prohibitions for a user before allowing access.

NetViews Policy Language and Maintenance: NetViews leverages NGAC's existing language and policy specification tools. The policy configuration can be set up or updated manually by an administrator or through an event processing module for a dynamic update. The events can trigger obligations and in turn dynamically update the policy. NGAC follows an administrative policy model consisting of administrative policy elements (e.g., admin rights, associations, prohibition, obligations, and routines). We exclude these administrative aspects for simplicity. Currently, both JSON and GUI-based graph specification is available. As the NGAC project evolves, NetViews will continue to benefit from new features and usability enhancements. In current setup, an administrator needs to identify subjects, access rights, and objects and write the specifications for generating a meaningful policy.

3.4.2 NetViews Identity Mapping Service

NetViews requires an external service to map the network packet 5-tuple information to real user-devices and server hosts. Usually the identity mapping is relatively static with respect to the rate of IP packets traversing the network. Therefore, identity mapping updates are not on the critical path for access control decisions.

NetViews's identity mapping service must coordinate with the identity services used by the enterprise. The most straightforward scenario is traditional enterprise network environments that use static DHCP to allocate IP addresses to known hosts. They also rely on MAC address-based allow lists to prevent MAC address spoofing. In environments that use 802.1x [76], the identity mapping service needs to coordinate with the authenticator and authentication server (e.g., RADIUS) to determine the policy user that corresponds with the supplicant. Additional security protection against IP and MAC address spoofing can be provided by enhanced solutions such as SECUREBINDER [74]. For environments with shared hosts that authenticate via a domain controller (e.g., Microsoft Active Directory), the identity mapping service can coordinate with the domain controller to associate hosts with the currently logged-in user.

Finally, when the mapping between IP address and policy identity is dynamic, the NetViews controller should be notified when a mapping is invalidated (e.g., logout). Ideally, logout events should cause the NetViews controller to remove corresponding forwarding rules. However, given sufficiently short forwarding rule idle timeouts (e.g., the ONOS default is 10s), allowing the rules to expire will provide sufficient security in most deployments.

3.4.3 Access Control Semantics

Unlike in OS access control, where enforcement is needed just in one direction of the communication (for example, enforcing access control on a process reading a file), network access control must allow network flows in *both directions* for successful network communication. Specifically, *stateless* firewalls (also known as stateless firewall filters and access control lists) must define rules that allow flows in both directions. They consider each packet in isolation, using Layer 3 and 4 header flags to differentiate the first packet in a connection from the reply traffic. In contrast, *stateful* firewalls only define rules for connection initiation and then track the connection to allow all subsequent packets in both directions. Stateful firewalls are generally viewed as more secure and have replaced nearly all stateless firewalls. The primary benefit of stateful firewalls is that they prevent many types of network scanning commonly used for reconnaissance before an attack, e.g., ACK scanning.

Supporting bi-directional network flows is at the root of a traditional challenge for network access control. Specifically, *stateless* firewalls (also known as stateless firewall filters and access control lists) must define rules that allow flows in both directions. Whereas *stateful* firewalls only define rules for connection initiation and then track the connection to allow all subse-

quent packets in both directions, a stateful firewall may maintain a TCP state machine for each connection, updating the state to open, open sent, synchronized, synchronization acknowledge, or established as the connection is established. In contrast, a stateless firewall considers each packet in isolation, using Layer 3 and 4 header flags to differentiate the first packet in a connection from the reply traffic: e.g., for TCP, a stateless firewall policy only allows the 5-tuple in the reverse direction if the TCP header contains an ACK flag. This parameter prevents the first packet of a new connection from matching the rule.

Stateful firewalls are generally viewed as more secure and have replaced nearly all stateless firewalls. The primary benefit of stateful firewalls is that they prevent many types of network scanning commonly used for reconnaissance before an attack. For example, if an attacker crafts a packet with the TCP ACK flag set, a stateless firewall will allow it (due to the rule for return traffic). Since the packet is not part of an active TCP connection, the target victim host will reply, informing the attacker of such. As a result, an attacker can discover the IP addresses of hosts behind the firewall.

NetViews Enforcement Semantics: NetViews assumes an SDN network configured with reactive forwarding. When a client c sends its first packet to a server s, the NetViews policy is consulted. If the packet is allowed, the NetViews SDN controller installs forwarding rules that match the 5-tuple network flows *for both directions*.[1] These forwarding rules are installed in all switches on the path between c and s. We note that NetViews relies on the underlying SDN controller (e.g., ONOS) and installed forwarding applications to determine the specific forwarding path.

Unlike stateful firewalls, NetViews does not track TCP connection state. However, installing 5-tuple matching forwarding rules for both directions provides equivalent security. The server s cannot send a packet to the client c before c initiates the connection. Switches on the forwarding path simply could not deliver the packet. Furthermore, for the above semantics, s can only send packets to the specific TCP port on c that initiated the connection. It can do so only for the lifetime of the forwarding rule, which will timeout after a predefined period (e.g., 10 seconds). Similarly, a malicious host m on the network cannot send packets to any other host without consulting the NetViews SDN controller.

Multi-Connection Optimization: SDN switches cannot efficiently manage a large number of forwarding rules. In an OpenFlow-based network, installing `FlowMod` rules for each 5-tuple flow may overflow the TCAM in the switches, causing increased `PacketIn` events and significantly degrading network performance. However, we observe that the above enforcement semantics are more strict than required. Instead of matching the specific client TCP port, NetViews can install forwarding rules that match *any* client TCP port. This change will significantly reduce the number of required forwarding rules for application protocols such as HTTP that require

[1] As discussed in Section 3.5, determining the forwarding rule for reply traffic for non-TCP packets requires some consideration.

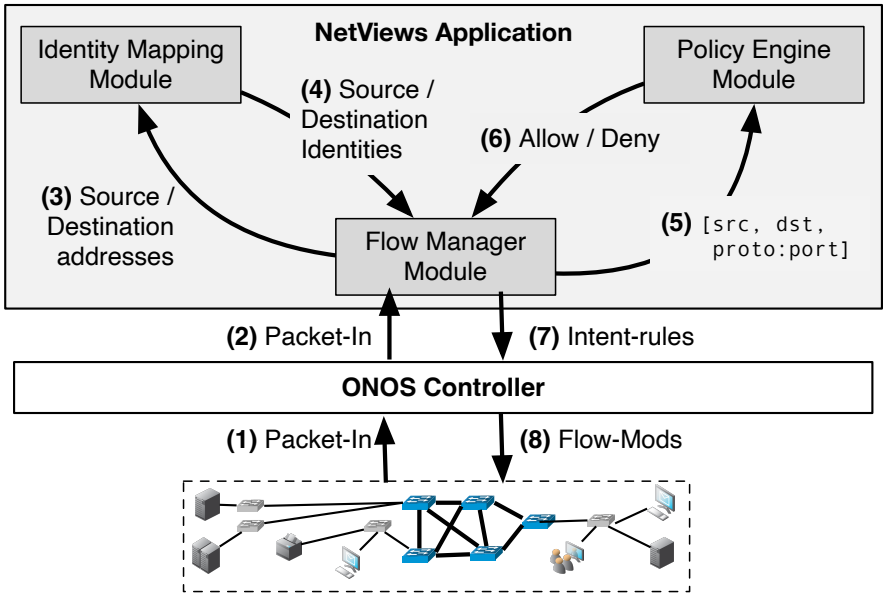


Figure 3.4: NetViews prototype implementation.

clients to make many TCP connections to the same server and port.

We observe that this performance optimization (not matching 5-tuple) does not have a significant negative impact on security. The only hosts that can exploit the more permissive forwarding rules are hosts to which connections have already been permitted. Furthermore, with reasonable forwarding rule timeouts, the risk is further reduced. Thus, NetViews's goal of controlling network visibility and preventing reconnaissance through network scanning is still achieved. In summary, the perimeter-less and least-privilege nature of NetViews eliminates the need for stateful tracking to prevent reconnaissance via network scanning.

3.5 NetViews Implementation

Our NetViews prototype implementation is built as an SDN application on top of ONOS version 2.3.0. It does not require any modification to the ONOS controller, which simplifies code maintenance and allows deployments into existing ONOS installations. As shown in Figure 3.4, the NetViews application consists of three logical components: the flow manager, the policy engine, and the identity mapper. While our implementation combines all functionality in one application, alternative implementations could easily decouple the three components to enhance scalability.

3.5.1 Flow Manager

OpenFlow reactive forwarding applications receive `PacketIn` events whenever a switch receives a packet that does not match any of its forwarding rules. The `PacketIn` event contains packet header information, including the Layer 2, 3, and 4 protocol types and identifiers. Forwarding applications commonly respond to `PacketIn` events by sending `FlowMod` messages to the originating switch. A `FlowMod` message contains a set of match criteria (e.g., source and destination IP) and an action (e.g., forward out physical port `3`). The switch uses the `FlowMod` message to update its forwarding rules.

Since the entire forwarding path can be determined at the time of the first `PacketIn` from the switch closest to the source, it is inefficient to wait for the `PacketIn` events from the subsequent switches on the forwarding path. As such, an SDN forwarding application can avoid additional delays for delivering the first packet by proactively sending `FlowMod` messages to the subsequent switches in hope of preventing additional `PacketIn` events. However, managing many different `FlowMod` rules for many switches can become very complex. To ease the development of applications, ONOS provides a forwarding primitive called an *Intent* [2], which provides a *one big switch* abstraction similar to Pyretic [108]. An ONOS Intent [2] defines match rules. ONOS compiles an Intent and manages all of the `FlowMod` messages for individual switches. Our implementation relies on ONOS's existing forwarding path algorithm.

Identifying Reverse Flow: For each authorization, NetViews installs one Intent for each direction of network traffic. Identifying the reverse flow for TCP connections is straightforward, as the TCP port information is symmetric. However, not all protocols use symmetric identifiers. For example, ICMP packets use type information (e.g., `ECHO`, `REPLY`) to indicate the direction of the flow. Since most firewalls drop nearly all ICMP types, NetViews currently only handles ICMP Ping messages, defining a match rule for `REPLY` for the reverse flow. In contrast, ARP packets must be handled differently. While ARP packets do not contain an IP header, they do include the source and destination IP address in the protocol address information. However, ARP spoofing attacks could allow an attacker to circumvent the policy. Fortunately, SDN controllers can mitigate ARP spoofing using a proxy ARP approach. That is, NetViews maintains a mapping between IP addresses and MAC addresses (e.g., from static DHCP configuration). When NetViews receives a `PacketIn` for an ARP request at the first switch, it consults the mapping and performs an access control check to determine if the ARP request is allowed by the policy. NetViews additionally 1) ensures the ARP request source MAC address matches the MAC address expected for the IP address, and 2) performs an access control check to determine if the ARP request is allowed by the policy.

Intent Installation: The Intent installation implementation was more subtle than initially expected. ONOS's Intent framework uses a `TrafficSelector` object that identifies a subset of network traffic based on packet header fields and patterns. The framework then compiles the

Intent into a set of `FlowRule` objects that are installed to switches on the path. A reactive Intent application creates Intents in response to `PacketIn` events. While the Intent is being compiled and installed, a similar `PacketIn` may occur. In order to avoid creating duplicate Intents, Intent applications deterministically create an Intent *key* based on packet characteristics. When a `PacketIn` that matches the key of an Intent pending installation, a `PacketOut` message is returned. A `PacketOut` message instructs the switch to forward a given packet but not update its forwarding rules. Many `PacketIn` events may occur before the Intent is fully installed.

We began by modeling ONOS's sample Intent reactive forwarding (`ifwd`) application. However, there are significant differences. First, `ifwd` only requires one Intent, because it matches packets using an empty `TrafficSelector`. In contrast, NetViews uses a non-empty `TrafficSelector` to match Layer 3 and 4 information and hence needs Intents for both directions.

The second key difference involves the creation of the Intent key and has subtle implications on handling `PacketIn` events that occur for the *reverse flow*. `ifwd` defines its Intent *key* by concatenating the device IDs in lexicographical order. As such, `PacketIn` events for the reverse flow will match the Intent key. In contrast, NetViews requires two Intents with fine-grained keys. The Intent key for the forward direction is defined by concatenating the source IP, source MAC, destination IP, destination MAC, protocol, and destination port. The Intent key for the reverse direction reorders these values, predicting the order of the reverse flow. The Intent key for the reverse flow also appends the value "`RETURN`" to distinguish it from a new flow, which would require an access control check. To quickly identify reverse flows, NetViews implements a local cache of return keys, releasing the packet with a `PacketOut` on cache hit.

Finally, the ONOS Intent framework does not support idle-timeout for the Intent itself. As a result, whenever a particular flow rule expires after an idle-timeout period (default, 10 seconds), the Intent API reinstalls the flow rule. Therefore, NetViews stores a timestamp for each installed Intent. It uses a separate thread to deactivate Intents after a pre-specified period (10 seconds in our implementation). Note that Intent deactivation does not trigger flow rules expiration, and therefore the packets will continue to be forwarded without consulting the NetViews controller as long as the client-server communication continues.

Note that both the ONOS Intent and `FlowMod` timeouts are an administrative trade-off between security and performance. A shorter idle-timeout will result in greater `PacketIn` events. A longer idle-timeout provides a compromised account or malicious insider more opportunity to connect to network resources. However, for most scenarios a timeout on the order of seconds is reasonable. For example, removing the access of a fired employee requires a policy administrator to manually change the policy, an action that may take on the order of tens of minutes in a typical organization.

3.5.2 Policy Engine

The policy engine module is based on the reference implementation of NGAC [75]. Specifically, we used the `policy-machine-core` project, which provides core components of the NIST Policy Machine. It includes APIs to manage NGAC Graphs (we use the JSON interface), query the access state of a graph, and explain why a user has permissions on a particular resource. There are four main packages in the core library: the Policy Information Point (PIP), the Policy Administration Point (PAP), the Event Processing Point (EPP), and the Policy Decision Point (PDP). The PIP package provides the necessary interfaces (and in-memory implementations) for managing an NGAC graph, prohibitions, and obligations. A combination of the PAP and PDP allows NetViews to request permission decisions using user identity, object identity, and permission-type.

3.5.3 Identity Mapping Service

Our NetViews implementation assumes a static identity mapping managed via a configuration file that is auto-generated after manual administrative actions. This scenario corresponds to static DHCP assignments, which are used by many enterprises and universities. The identity mapping module loads a simple configuration file to populate a data structure that maps a $\langle IP, MAC \rangle$ pair to a unique $\langle user, device \rangle$ pair for users and a unique ID for servers. As discussed in Section 3.4.2, more dynamic scenarios that use 802.1x authentication (e.g., WPA Enterprise) requires a communication interface with RADIUS and DHCP servers. To minimize first-packet latency in dynamic scenarios, identity maps should be cached locally and invalidated on DHCP release. We leave the implementation of a dynamic identity mapping service to future work.

3.6 Security Analysis

NetViews significantly reduces an attacker's ability to move laterally within a network, even if it has compromised user credentials. This section provides a concrete demonstration of this benefit using our reference topology in Figure 3.1. Specifically, we use reachability-based attack graphs proposed by Lippmann et al. [95, 133] to validate network defense [116]. These works use firewall configuration and host vulnerability information to build attack graphs that describe how far an attacker can progress through a network. Our analysis assumes all hosts are vulnerable, which corresponds to the motivating NotPetya and Solorigate attacks described in Section 3.2.

Lippmann et al. [95] define three key concepts. First, a *reachability matrix* R is defined using outbound interfaces on all hosts i as rows and all active hosts j as columns. $R[i, j]$ is $true$ if a logical connection is possible between source i and destination j; otherwise, $R[i, j]$ is $false$. Second, an *attack graph* is a directed graph $G = (V, E)$ that shows how far attackers can

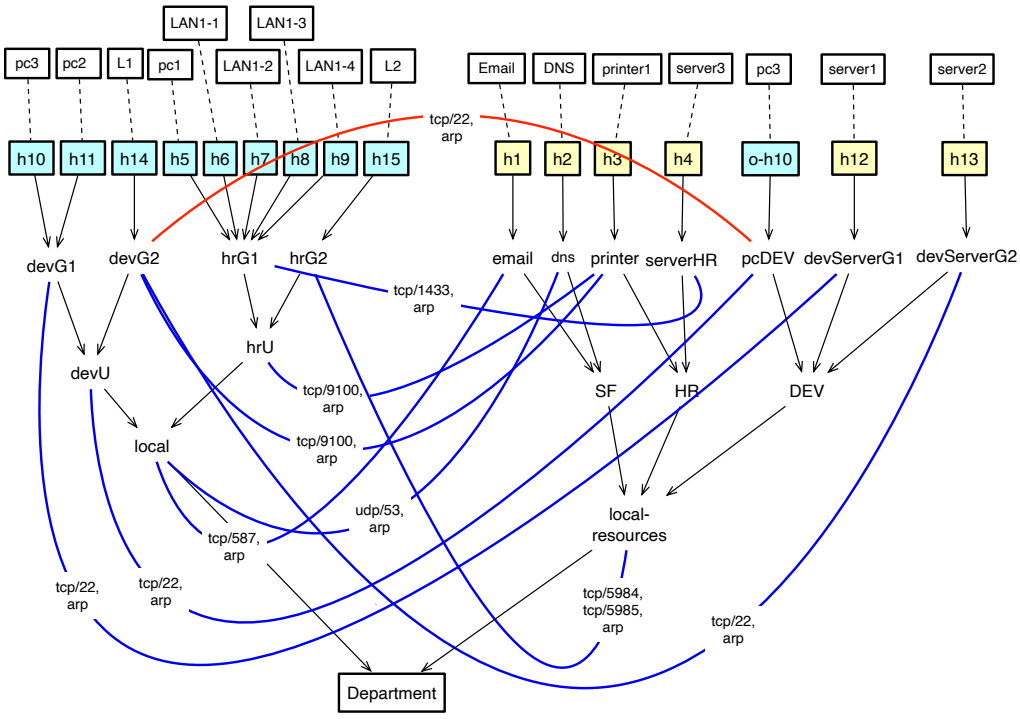


Figure 3.5: NetViews policy for reference topology of Figure 3.1

progress through a network by successively compromising exposed and vulnerable hosts. The vertices V represent network hosts. For hosts $v_i, v_j \in V$, there exists an edge (v_i, v_j) when host v_i can make a logical connection to v_j. Finally, a *predictive attack graph* is a directed acyclic graph (DAG) rooted at a specific source host (assumed to be compromised) of interest. A new edge/host pair is added only if this pair is not already attached to the root of the graph or to any node along the path from the root to the current source node. The predictive attack graph is constructed using a breadth-first search traversal of the reachability matrix from the host of interest, only adding a new edge if the target host is not already included.

Analysis setup: Using the reference topology in Figure 3.1, we defined a NetViews policy (Figure 3.5) and a corresponding traditional stateful firewall policy (Table 3.1) assuming the network segments shown in the figure. Given the size of the topology, we hand-generated the reachability matrix, which is not uncommon [95, 133]. Note that while the reachability matrix generation traditionally considers if a host has a vulnerability, our analysis assumes a NotPetya or Solorigate scenario where all hosts have vulnerabilities, as discussed above. We then wrote a script to generate a predictive attack graph from the reachability matrix following Lippmann et al.'s algorithm [95]. We generated predictive attack graphs for each host for each of the two policy models.

Results: Figure 3.6 shows an example of the substantial reduction in the predictive attack

Table 3.1: Firewall policy reflecting the NetViews policy in Figure 3.5

	Decision	Source IP	Source Port	Destination IP	Destination Port	Protocol
Server Farm						
	allow	10.0.100.*	any	10.0.100.2	53	udp
	allow	10.0.100.*	any	10.0.100.1	587	tcp
	allow	10.0.100.15	any	10.0.100.1	5985	tcp
	allow	10.0.100.15	any	10.0.100.1	5984	tcp
	allow	10.0.100.15	any	10.0.100.2	5985	tcp
	allow	10.0.100.15	any	10.0.100.2	5984	tcp
	deny	any	any	any	any	any
HR Floor						
	allow	10.0.100.14	any	10.0.100.3	9100	tcp
	allow	10.0.100.5	any	any	any	tcp
	allow	10.0.100.6	any	any	any	tcp
	allow	10.0.100.7	any	any	any	tcp
	allow	10.0.100.8	any	any	any	tcp
	allow	10.0.100.9	any	any	any	tcp
	allow	10.0.100.15	any	any	any	tcp
	deny	any	any	any	any	any
Developer Floor						
	allow	10.0.100.10	any	any	any	tcp
	allow	10.0.100.11	any	any	any	tcp
	allow	10.0.100.12	any	any	any	tcp
	allow	10.0.100.13	any	any	any	tcp
	allow	10.0.100.14	any	any	any	tcp
	allow	10.0.100.15	any	10.0.100.12	5984	tcp
	allow	10.0.100.15	any	10.0.100.12	5985	tcp
	allow	10.0.100.15	any	10.0.100.13	5984	tcp
	allow	10.0.100.15	any	10.0.100.13	5985	tcp
	deny	any	any	any	any	any

graph for NetViews over a traditional segmented network policy enforcement with firewalls. While this figure demonstrates the qualitative benefit of NetViews, it does not quantitatively describe the overall benefit to the network. Table 3.2 focuses on the reachability of the three high-value servers in the reference topology (Figure 3.1). The table enumerates the number of hosts that can reach the server for a specific "hop-count." That is, a hop-count of one indicates hosts can access the server directly. A hop-count of two indicates a host has to compromise one other host before accessing the server. The table shows the number of hosts for hop-counts

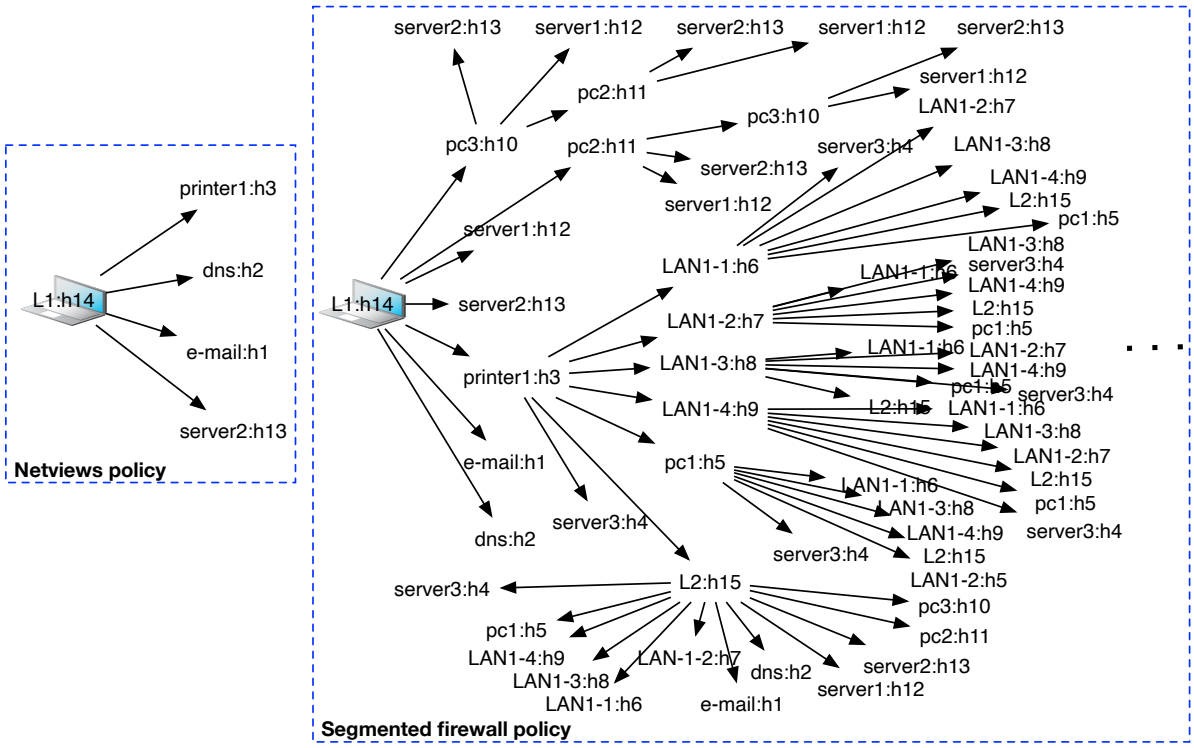


Figure 3.6: Attack graph visualizing possible reconnaissance and lateral movement from a compromised host. The graph depicts how far an attacker compromising user *Alice* can progress within the reference network from Figure 3.1.

up to five. Notably, NetViews nearly eliminates the value of lateral movement for an attacker. The table does show that `server1` has two hosts with a hop-count of two. These links results because SSH access is allowed to a development host `pc3`. Thus, the benefit of a NetViews is dependent on the policy, which is to be expected. However, NetViews offers significant potential to drastically reduce lateral movement.

3.7 Performance Evaluation

This section evaluates the performance of NetViews and the Policy Engine, answering the following questions.

Q1 How does NetViews compare with other ONOS forwarding applications in terms of latency and throughput?

Q2 How does NetViews scale with the number of flows?

Q3 How does the Policy Engine scale for more complex policies (and topologies)?

Table 3.2: Number of hosts reachable in hop-counts 1 to 5 for the reference topology (Figure 3.1) based on policy type

Policy Type	hop-count	server1	server2	server3
NetViews	1	3	2	6
	2	2	0	0
	3	0	0	0
	4	0	0	0
	5	0	0	0
Segmented Firewall	1	4	4	8
	2	9	9	10
	3	9	9	10
	4	9	9	10
	5	9	9	10

Table 3.3: Description of Evaluation Topologies

Topology	Devices	Switches	Details
Reference	13	12	Topology from Figure 3.1
Cisco [155]	12	10	Cisco enterprise network
MiniStanford [155]	100	25	Stanford backbone network

3.7.1 Experimental Setup

We consider three example enterprise network topologies, measuring network performance of three ONOS applications.

- **Baseline (fwd)**: This scenario uses the ONOS Reactive Forwarding application (`org.onosproject.fwd`) available with the ONOS distribution. The application will receive a `PacketIn` request, *allow* all communication between source and destination, and install corresponding `Flow Rules`. We consider this scenario as it is the most basic way to maintain networking functionality in an SDN environment without any policy enforcement.
- **Intent Forwarding (ifwd)**: This scenario uses the Intent Forwarding application provided by the ONOS sample library. It also *allows* all communication between source and destination on receiving a `PacketIn` request and installs corresponding Intents.
- **NetViews implementation (NetViews)**: This scenario is our implementation, and includes all the system components discussed in Section 3.5 (see Figure 3.4).

Table 3.3 summarizes our three representative enterprise topologies showed in Figure 3.7. The Reference topology was introduced in Figure 3.1; we used this simple topology to introduce NetViews design parameters. The Cisco and MiniStanford were previously used in other evaluations (e.g., PSI [155], HeaderSpaceAnalysis [80], Resonance [113]) and represent topologies provided by Cisco and Stanford, respectively. Each link in the topology is Gigabit Ethernet.

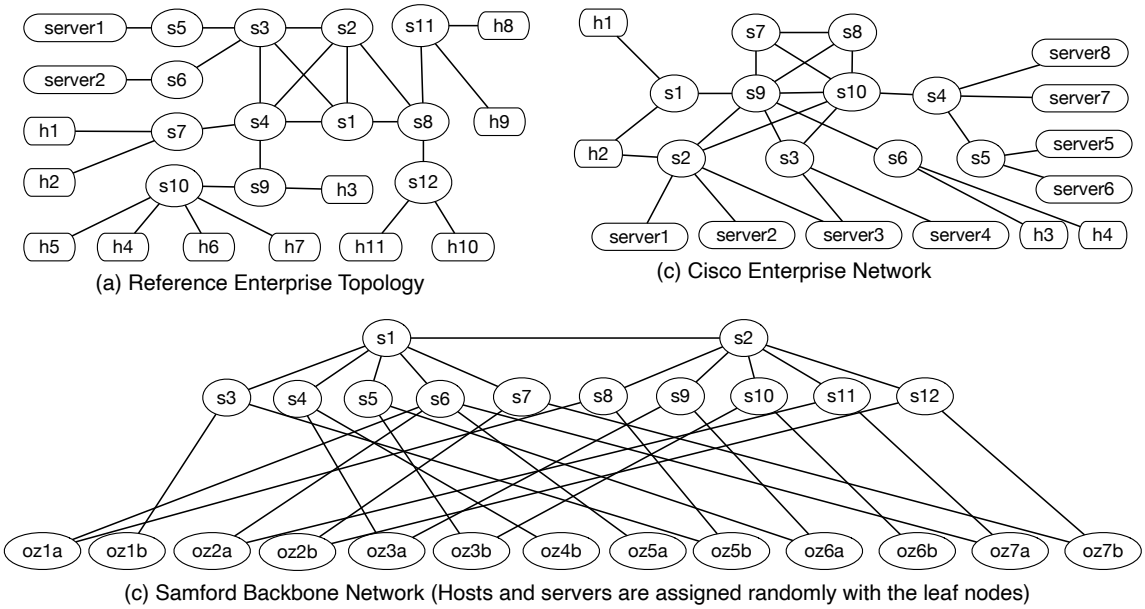


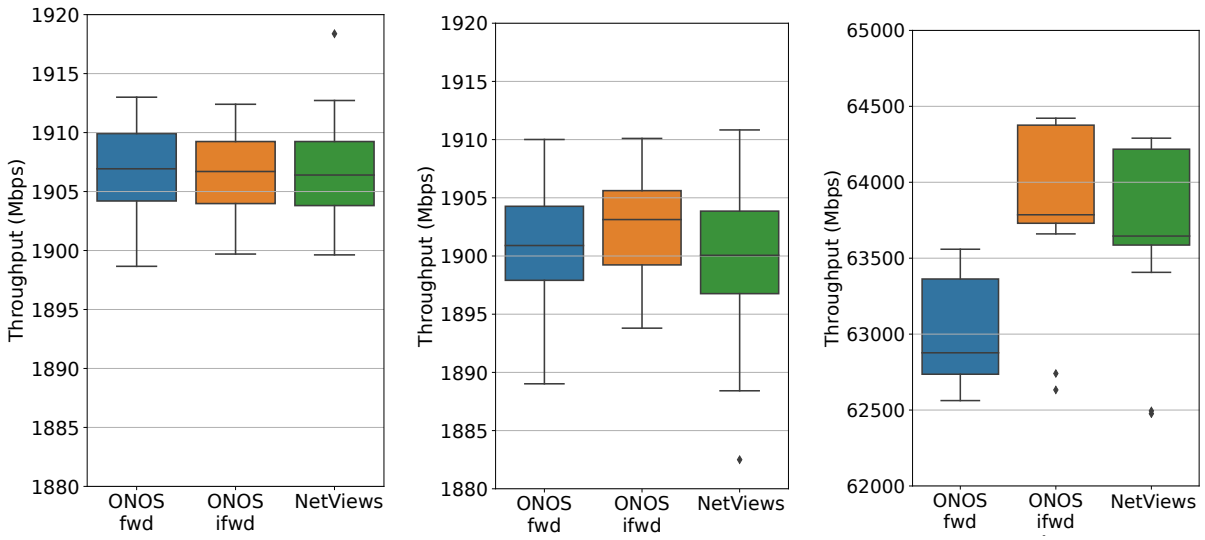
Figure 3.7: Evaluation Topologies

We implemented each of these topologies in Mininet [140]. Each measurement was taken on a Ubuntu 20.04 LTS (Linux Kernel 5.4.0) VM in QEMU with 16 Cores and 32 GB of RAM. These VMs are spawned from a server with 2 Intel Xeon Silver 4114 CPUs (20 physical cores at 2.20 GHz).

To evaluate throughput, we used the standard tool iPerf3 [40] (version 3.7). This approach allows us to test network throughput under maximum load (all hosts connected to all servers), as well as scale the amount of connections by increasing the number of parallel streams for our scalability analysis. To evaluate latency (initial packet, and excluding the initial packet) we use MTR [24] (version 0.93). As the complexity of policy does not impact performance, we use a policy that allows all connections. Each experiment lasts 60 seconds and is repeated 50 times. We present results as an arithmetic mean over these 50 runs.

3.7.2 Performance Overhead

Our principal concern in this section is to determine the overhead introduced by NetViews compared with baseline ONOS applications. Because each network is unique and there is no "typical" level of traffic, we choose to compare throughput in the *worst case scenario*, where all "subject" hosts are attempting to send as much data as possible to the "object" servers in each topology simultaneously. To evaluate the marginal latency of adding new flows to the network all "subject" hosts start MTR pings to the "object" servers sequentially, with 1 second between each new flow.



(a) Cisco **(b)** Reference **(c)** MiniStanford

Figure 3.8: Aggregate throughput for different topologies (scales differ for readability)

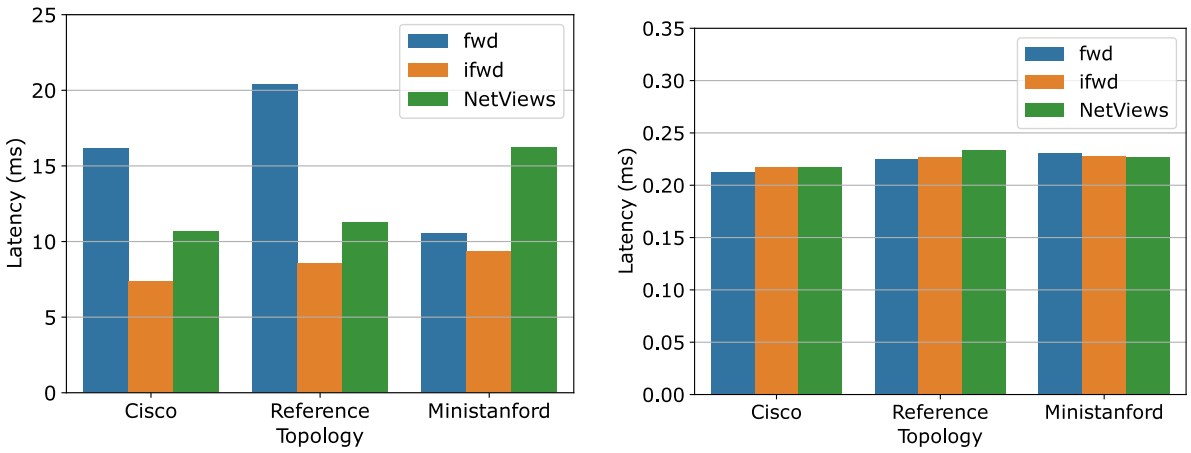
Throughput

We compared the average throughput of NetViews to baseline ONOS apps. Because the throughput seen by any individual host is a function of topology and transport layer fairness, we characterize the total *aggregate* throughput of each topology by summing the throughput of all individual flows.

Figure 3.8 shows boxplots indicating the distribution of observed throughputs for each experiment run for each combination of ONOS app and topology. While each topology sees a different magnitude of throughput, there is considerable overlap between the three applications. Specifically, in the Cisco and Reference topologies, the median NetViews throughput falls well within the the inter-quartile range of the baseline cases and the change in medians is well under 1%. Moreover, in the scaled MiniStanford topology, we find the median aggregate throughput to be well above that of the `fwd` baseline application, and within approximately 200 *Mbps* of the `ifwd` baseline application. In Figure 3.8a the median aggregate throughput hovers between 1905 and 1908 *Mbps*. In Figure 3.8b the median aggregate throughput falls right around 1900 and 1905 *Mbps*. Finally, in Figure 3.8c the median aggregate throughput for `ifwd` and NetViews falls between 63550 and 63750 *Mbps*, while for `fwd` it sits around 62900 *Mbps*. We conclude that NetViews does not show any significant throughput overhead over the `fwd` or `ifwd` applications.

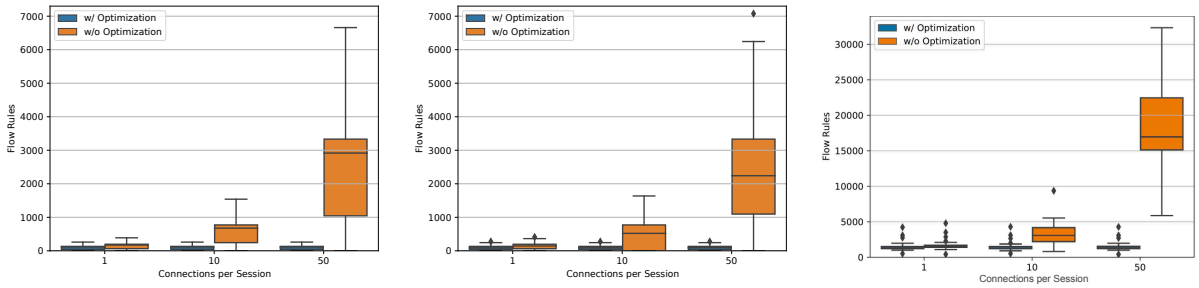
Latency

SDN reactive-forwarding applications experience a greater latency for the first packet. Thus, we evaluate both *initial latency* (i.e., the latency of packets causing `PacketIn` events) and n^{th}-



(a) Average initial packet latency **(b)** Average n^{th}-packet latency

Figure 3.9: Average latency for different topologies (scales differ for readability)



(a) Cisco **(b)** Reference **(c)** MiniStanford

Figure 3.10: Number of Flow Rules in switches for NetViews with and without the optimization (scales differ for readability)

packet latency, (i.e., the latency of all subsequent packets). We ensure a clean start (no Intents and no `Flow Rules` already installed) at the beginning of the experiments to ensure we capture `PacketIn` events in each measurement.

Figure 3.9 shows that NetViews has acceptable latency compared to `ifwd` for both the initial and n^{th}-packet. Figure 3.9a shows our initial packet latency results. In the Cisco topology we see a 3.286 ms (44.5%) latency increase for NetViews compared to `ifwd`, in the Reference topology a 2.687 ms (31.4%), and in the MiniStanford topology a 6.868 ms (73.5%) increase. Both `ifwd` and NetViews outperform `fwd`. Figure 3.9b shows our n^{th}-packet latency results. For the Cisco topology we see a negligible difference in latency (<0.01%), in the Reference topology a 0.0058 ms (2.59%) increase, and in the MiniStanford topology a 0.0007 ms (0.3%) decrease.

3.7.3 Multi-Connection Optimization

The multi-connection optimization reduces the number of `PacketIn` events and hence the number `Flow Rules` stored in the TCAM of switches. It provides the greatest benefit when clients establish many connections to a server as part of the same session. Specifically, for each connection in the session, the client uses a different ephemeral source port to the same server port. To characterize the benefit of the optimization, we emulate sessions with different numbers of parallel connections, measuring the resulting number of `Flow Rules` with and without the optimization. In Figure 3.10, the x-axis shows the number of parallel connections for the session, and the y-axis shows the average number of `Flow Rules` installed in switches for different topologies. The boxplots illustrate the distribution of `Flow Rule` counts across all switches in each topology.

The first set of boxes in all three figures depicts the scenario with no parallel connections, which results in policy enforcement based on 5-tuple packet information without the optimization. There is no match on the client port with the optimization (to allow ephemeral client ports). Even without parallel connections, the number of flow rules is higher without optimization. For example, in the case of the Cisco topology in Figure 3.10a, the median number of flows for the setting without optimization is 172. In contrast, with the optimization, that number goes down to 116.

The boxplots for 10 and 50 connections per session demonstrate where we expect the overhead reduction of the multi-connection optimization. The boxplot for the Cisco topology shows that for ten connections per session, the median number of `Flow Rules` per switch without the optimization increases to 772 from 172. In contrast, with the optimization, it remains at 116. Finally, with 50 connections per session, the number of `Flow Rules` per switch without optimization is 2,916, while with optimization, it remains at 116. A similar trend emerges for the other topologies. These results demonstrate that the multi-connection optimization can result in significantly fewer `Flow Rules`.

3.7.4 Performance of Policy Engine

To understand the impact of the Policy Engine on the overall NetViews overhead, we analyzed the response time of *policy-machine-core* using random policy graphs as in [16, 102]. We used the graph generation algorithm from Basnet et al. [16].

We let the number of user and object nodes be defined from node count n, $u = \alpha \times n$ and $o = (1 - \alpha) \times n$. For each pair of $\langle u_i, o_i \rangle$, we first created two separate single-rooted binary tree structures using only user attribute ua and object attribute oa nodes, respectively. The number of ua and oa nodes depend on the predefined policy tree height h. For each tree with height h, we generate $2^{h+1} - 1$ number of oa and ua nodes. After the tree creation, the u_i node is made the new root of the user attribute tree and o_i of object attribute tree. These trees represent

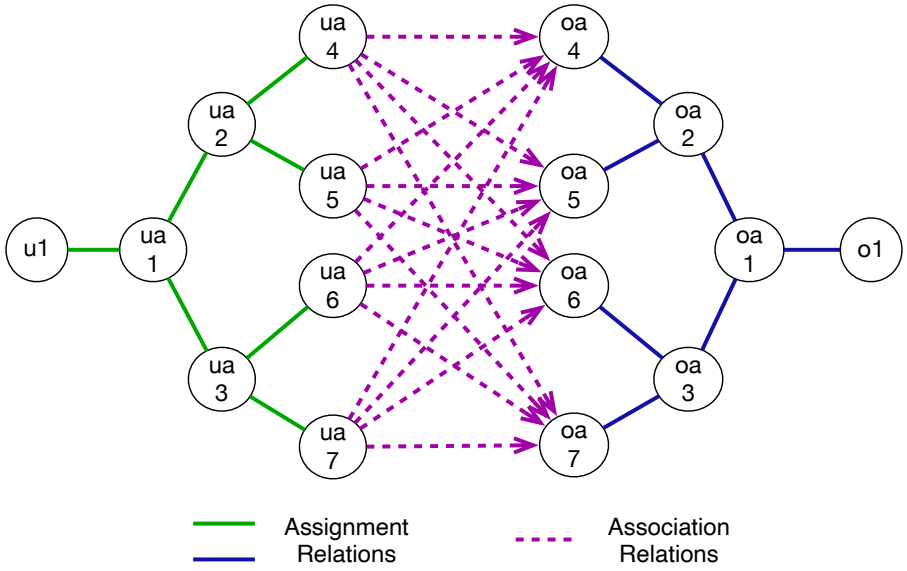


Figure 3.11: A subgraph for the pair of $\langle u_1, o_1 \rangle$ with height $h = 2$

the *assignment* relation of the policy graph. The aforementioned user attribute tree and object attribute tree of pair $\langle u_i, o_i \rangle$ is connected using the *association* relations, or edges, as shown in Figure 3.11. The idea of separate operation node creation from Basnet et al. [16] is omitted in our algorithm to reduce the graph generation cost. We map each leaf level *ua* node to each leaf level *oa* node, thus creating a total of $|ua_{leaf}| \times |oa_{leaf}|$ association arcs. We have used a set of δ permissions, and assigned each association arc with a permission label picked randomly from δ.

For this analysis, we generated access control graphs that varied in size from 4,000 to 1,280,000 vertexes (u, o, ua, and oa nodes) for different heights of the policy graph. However, the user (u), object (o) and height h are the main controlling parameters in our graph generation algorithm. As reported in Figure 3.12, we varied the total number of u and o nodes from 1,000 to 20,000 nodes ($\alpha = 0.6$), and h from 1 to 5 for each amount of total nodes. We considered a small permission set of $\delta = 10$ for measuring the decision time analysis. For each policy graph, we randomly picked 2000 permission requests from the set δ, and logged decision fetching delay from *policy-machine-core*. The average decision fetching delay is reported in micro-seconds.

Figure 3.12 indicates that the height of the policy graph is the main contributory factor to the decision delay. However, overall average delay is minimal, even for the 20,000 node (u and o) benchmark, with 1,280,000 graph vertexes (u, o, ua, and oa). The delay ranges from approximately 100 μs for height of one and 1250 μs for height of five. Therefore, the policy engine does not significantly affect overall operational performance of an enterprise network.

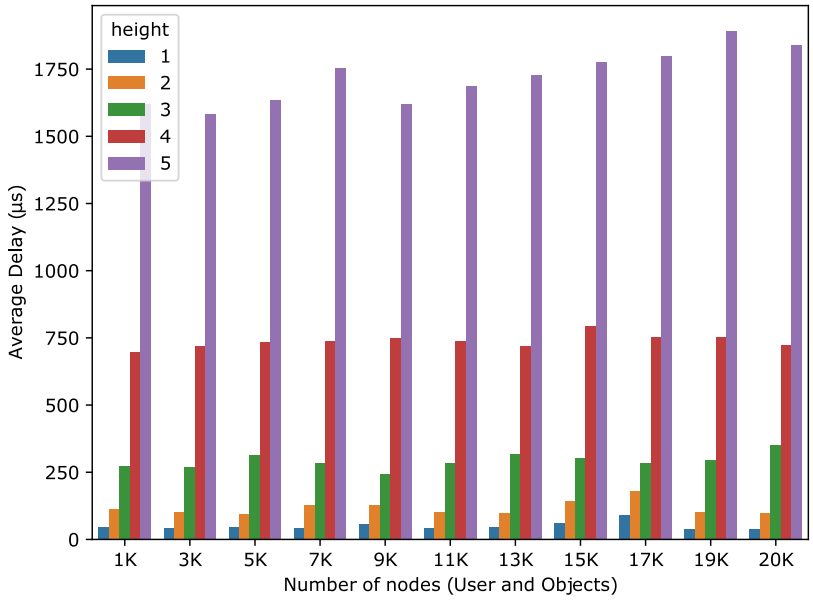


Figure 3.12: Average response time of policy-machine-core using random policy graphs

3.8 Discussion and Future Work

Scalability: We showed negligible overheads for latency and throughput on test networks from the literature, indicating our approach will scale equivalently to standard Intent Forwarding. We are limited, however, by publicly available datasets and topologies to test more extensive networks. The policy engine can exist either within the NetViews application or as an external server. If the policy engine is external, NetViews should implement a caching protocol within the SDN application. The external policy engine server should invalidate the cache when the policy changes. Additionally, ONOS supports multiple distributed SDN controllers to enhance scalability. We leave the investigation of policy cache management and multiple controllers to future work.

Policy Generation and Maintenance: NGAC can express a range of access control models, including both RBAC and ABAC, which are extensively used by enterprises for non-network access control. An administrator's first step for creating a policy is identifying subjects, access rights, and objects. Our current policy must be specified manually. We anticipate the development of tools to help automate the process. For example, a tool could extract policy from an enterprise's existing Active Directory installation. Our current implementation also requires the policy engine to reload the policy whenever there is a policy change. This is reasonable, because policy changes are infrequent relative to policy queries. In future work, we are exploring how to handle ephemeral and dynamic policy changes using NGAC's obligation primitive.

Deployment and Scope: NetViews seeks to provide access control within an enterprise's on-premises network environment. As enterprises move business applications into the cloud, an application-based Zero Trust model such as BeyondCorp may be more appropriate to protect those applications. That said, NetViews can also provide value *within* a cloud network environment; this is a topic of future work. As discussed in Section 3.2 , even if an enterprise moves all of their business applications to the cloud, they still require enhanced protections for their on-premises network environment. NetViews needs to be adapted to a given enterprise environment. For example, as noted in Section 3.5 our current implementation requires adopters to coordinate with 802.1x infrastructure. An enterprise network also may not be fully SDN-capable. While SDN-capable access switches and WiFi access points are ideal, legacy switches with VLAN capabilities can be used to isolate traffic at the access switch layer. Similar capabilities can be achieved by configuring WiFi access points to use isolation mode. In both cases, network traffic is shunted up to an SDN-capable distribution switch. However, even without changes to access switch and access point configuration, NetViews can provide meaningful value as it will opportunistically mediate any traffic that reaches the distribution switch. Such a deployment would still providing better security than existing perimeter-based defenses, even if it allows small pockets of unmediated network traffic. This approach also provides a path for incremental deployment.

3.9 Summary

While application-based Zero Trust architectures help enterprises secure their business applications by moving them to the cloud, they ignore the importance of securing the on-premises network environment that remains. This chapter has introduced a novel paradigm for enterprise network security called *Network Views* where each host has a different "view" of what other hosts and services exist in the network. This fine-grained least-privilege approach to network access control can significantly reduce lateral movement by attackers, even if user credentials have been compromised. NetViews builds on NIST's Next Generation Access Control to provide a dynamic and scalable policy model that supports the needs of large enterprises. We propose a multi-connection optimization that eliminates unnecessary first-packet latencies and significantly reduces TCAM requirements for SDN switches. Our NetViews implementation has latency and throughput comparable to reactive forwarding baselines. The source code of NetViews implementation is available in Git [8]. As such, NetViews provides a practical primitive for dissolving security perimeters within enterprise networks.

Chapter 4

Multisite NetViews

In Chapter 3, we focused on defining a Zero Trust network for on-premise devices through least-privilege network access control where each host has a different, limited view of the other hosts and services within a network. This chapter presents our subsequent work, MultiSite NetViews (MSNetViews), which extends a single, globally-defined enterprise network security policy to many geographically distributed sites.

4.1 Introduction

NetViews [9] is the most recent work in a series of research proposals [27, 113, 77] that use reactive software-defined networking (SDN) to enforce least-privilege, per-request connections between hosts within an enterprise network. These solutions for network access control assume the enterprise network exists in a single geographic site. However, most enterprises consist of many geographically distributed sites. Some enterprises have a small number of very large sites (e.g., the IBM topology in topology-zoo [118], has 18 sites, each with a massive number of resources and clients), while others have a large number of very small sites (e.g., Well's Fargo has 5300 branches across the USA). Employees also commonly move between sites and need differentiated access based on their specific location context. Globally managing user context across an organization is important. For example, Memo M-22-09 [119] states that "Zero trust architectures require metadata about the user to allow agencies to make risk-based decisions at the policy enforcement point. That metadata is maintained, updated, and supplied by systems that manage user identities, keeping the appropriate metadata associated with the correct user *even if that user leaves the organization or moves to a new position within it.* ... Using *centrally managed* systems to provide enterprise identity and *access management* services reduces the burden on agency staff to manage individual accounts and credentials." (emphasis added).

In this chapter, we present MultiSite NetViews (MSNetViews), which extends a single, globally-defined, enterprise network security policy to many geographically distributed sites.

MSNetViews extends prior work by managing many independently-operating reactive-SDN networks that dynamically react to employee movement between sites, enforcing both role-and location-based access control policies. MSNetViews also introduces the concept of site-specific "policy slices," which avoid unnecessary policy updates and limits the security policy available at each site on a "need-to-know" basis. Our performance evaluation shows that for an enterprise with sites globally distributed, the average time for policy state to settle after a user roams to a new site is well below two seconds, which is negligible with respect to the overall experience of traveling, often hours, and authenticating to a new network.

We make the following contributions in this work:

- *We propose the MSNetViews policy model for supporting user roaming between sites.* Location updates do not change the global policy specification, but rather parameterize the policy state at individual sites.

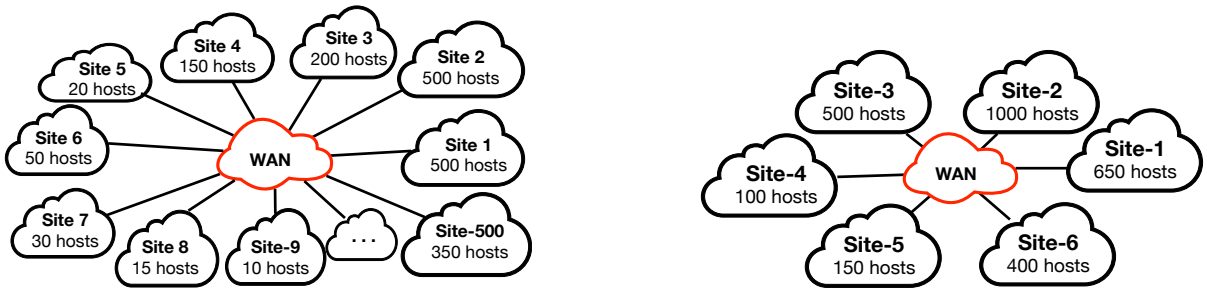
- *We propose* policy slicing *to minimize policy information loss when a site is compromised.* Each site maintains a site-specific policy containing only the policy elements and rules needed to govern access to local resources.

- *We propose a global administrative model that maintains the correctness of administrative updates.* With multiple administrators specifying policy for their individual sites, novel policy checks ensure mistakes do not cause interference between the role- and location-based policies.

Throughout the chapter, we assume all enterprise network sites are fully provisioned with reactive SDN technology such as OpenFlow. However, hybrid designs can achieve similar protection by using legacy VLAN-capable switches to shunt traffic from individual hosts to SDN-capable distribution switches [5, 91]. While both approaches require changes to networking infrastructure, transition plans are needed for resources that cannot practically be moved to cloud servers, particularly for US government organizations needing to meet the fiscal year 2024 deadline set by Memo M-22-09 [119].

Finally, similar to NetViews, MSNetViews is not a complete zero trust solution for on-premises devices. The goal of MSNetViews is to extend NetViews to a multisite setting. A complete zero trust solution should include device attestation and the behavioral analytics. We discuss MSNetViews and zero trust in Section 4.9.

Availability: The source code for our MSNetViews implementation is available at `https://github.com/netviews/ms-netviews`.



(a) Bank **(b)** Big-Tech

Figure 4.1: Example Enterprise Network Scenarios

4.2 Motivation

MSNetViews is motivated by the current threat models for enterprise and government networks, where systems can not be secure behind a moat-and-gate (e.g., firewall). Additionally, multisite enterprises intensify the challenge with multidimensional trust-base and functional requirements. The combination of least privilege policy and reactive SDN has the potential to ensure a secure functionality [9]. NetViews [9] leverage NGAC to define the access control policies that determine whether or not a given host can access another host through a specific transport layer ports in a single site enterprise. NetViews enforces policy using the ONOS SDN controller for OpenFlow.

NetViews and all other prior solutions assume a single network environment. However, many enterprises consist of multiple geographically distributed networks. Figure 4.1 shows two examples near the ends of the spectrum ranging from a bank with many small sites (e.g., Well's Fargo has 5300 branches across the USA) to big-tech enterprises with a few large sites (e.g., the IBM topology in topology-zoo [118], has 18 sites, each with a massive number of resources and clients). MSNetViews extends NetViews to multisite environments, allowing network administrators to specify and enforce a single, global, mandatory network policy across the entire enterprise.

Network security is critical for all enterprise or government network geographic sites. Following the ZTA principle, user identity should be the decision point for access control. However, identity-based least privilege access control is more challenging as appropriate contexts must be propagated across the sites. Unless maintained diligently, this can impose a huge security risk to the enterprise (e.g., impersonation, DDoS). Furthermore, determining the balance between functionality and security is critical with a disputable trust base. For scenarios like *Bank*, a small malicious or uninformed site has the potential to jeopardize the security of the entire enterprise. Additionally, a multisite environment is more dynamic in functionality and management.

Threat Model & Assumptions: We assume attackers have control of one or more hosts

within the enterprise and are knowledgeable about the environment and defenses. The attacker seeks to compromise devices, exfiltrate data, impersonate users, or disable enterprise services. We assume a worst case scenario where attackers have knowledge of remote code exploits for hosts (e.g., as in NotPetya [13]) and seek to mitigate movement using mandatory controls within the network. Furthermore, for organizations with many small sites, we assume a specific local site can be compromised, leaking enterprise policy and configurations of that site.

Our trusted computing base (TCB) includes the SDN security application, data plane devices, policy engines, and identity management services. We assume administrators securely access the policy configuration interface using TLS-protected communication and multi-factor authentication. We do not consider malicious-but-trusted administrators reconfiguring or updating their authorized system components. The TCB extends to other SDN applications running on the controller not separated from MSNetViews. We assume defenses for known DoS concerns for SDN are deployed.

4.3 MSNetViews Overview

Enterprise networks require least-privilege policies that restrict network communication between on-premises hosts. Extending prior solutions to consider multisite environments requires overcoming the following research challenges.

- *Users commonly move between sites, requiring differentiated access based on their location.* The policy and enforcement must dynamically update based on a user's location, and ensure that state is consistent across sites.
- *Compromise of a single site should not leak the global policy.* Security policies are often confidential. An exposed policy at one site should not leak policy details of unrelated sites.
- *Site administrators should only modify policies for their local resources.* Updates to the global policy should be controlled and maintain policy semantics.

We overcame these challenges using a global-local design that simplifies state consistency maintenance. MSNetViews consists of a global manager that serves as a permanent, central leader, which communicates policy and location updates to independent reactive SDN networks. As a result, MSNetViews never requires policies to be merged, eliminating the potential for policy conflicts. Using independent SDN networks also allows internal traffic to continue if the WAN connection fails. In addition to the global-local design, MSNetViews models location policy in a way that allows it to be parameterized, which eliminates full policy updates on each location update event, leading to faster consistency across sites.

Figure 4.2 shows how MSNetViews operates at both the global and local scales. The global manager coordinates policy management between sites. We envision the global manager residing

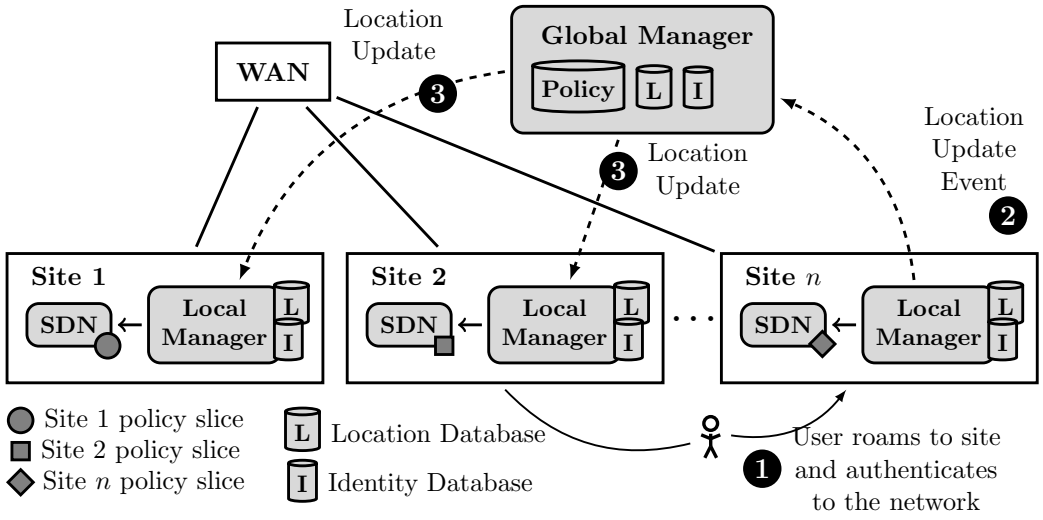


Figure 4.2: Overview of our approach. Each site runs its own SDN network with a unique subset of the global policy. Users roaming between sites cause location update events that are propagated to other sites as appropriate. Each SDN controller has a different access control policy, as indicated by the different shapes (circle, square, diamond).

in a cloud service, though it could be hosted within a site. Site administrators specify their network access control policy directly in the global manager through an administrator console. The global manager needs only to reestablish policy consistency across sites when a policy administrator updates the policy.

At the local scale, each site runs its own independent reactive SDN network with its own SDN controller and local manager to enforce policy. A local manager operates at each site, managing the local policy state, users, and interactions with the global manager. A user's physical movement initiate location and identity update-event, which initiate local policy update in the new-site dynamically. The global manager only propagates the location and identity information to other local-sites, allowing dynamic update in their local policy accordingly.

Modeling Location Policy: Shown in Figure 4.3, MSNetViews models location-specific policy by creating a separate *policy class* with separate user and object attributes. This separation (a) eases policy management by clarifying the two types of policies and (b) ensures the policy semantics remains intact. To demonstrate the importance for policy semantics, consider the case where both policy classes use object attribute oa, that is, $oa \rightsquigarrow pc_{role}$ and $oa \rightsquigarrow pc_{location}$. Any user that satisfies the location requirement ($u \rightsquigarrow ua$, $ua \rightsquigarrow pc_{location}$, and (ua, AR, oa)) can then access any object o, when $o \rightsquigarrow oa$, regardless of the "Role" policy.

We simplify policy management by not including explicit assignments from users to site-attributes in the global policy. Instead, we use NGAC obligations to create the dynamic assignments between user-device pairs (NGAC users) and user attributes for the "Location" policy class in the local policy. For example, the dashed assignment from $\langle Alice, L1 \rangle$ to S_2 in Figure 4.3

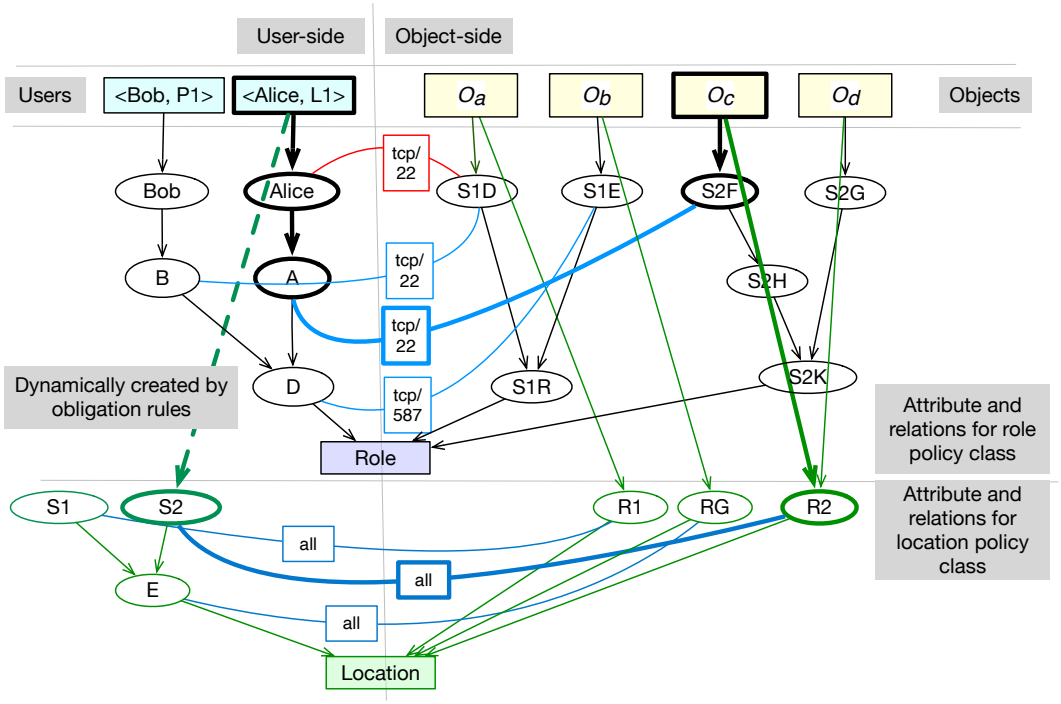


Figure 4.3: MSNetViews policy example depicting two sites ($S1$ and $S2$). Here, the user $\langle Alice, L1\rangle$ can SSH to object O_c while attached to user attribute $S2$. The bold lined paths through two policy classes ("Role" and "Location") indicates the access control paths needed to be followed for determining this "allow" decision. Downward blue arcs denote associations, each of which is annotated with a set of rights (e.g., *tcp/22*).

will be dynamically added in the local sites.

Supporting User Roaming: Users may move between sites (with a travel time ranging from several minutes to days), a process we call *roaming*. A user at a particular site may expect to be able to access local resources (e.g., printers and wireless displays). Conversely, they should *not* be able to access those local resources while not at that site. Other resources (e.g., hardened internal servers) may only be accessible from specific sites.

We present an overview of the roaming process in Figure 4.2. When the user authenticates to access a new site (e.g., via 802.1x in WiFi enterprise) (step ❶), MSNetViews triggers an event to modify *only* the local policy state via NGAC obligations. The local manager then informs the global manager (step ❷) of the location update event. The global manager, in turn, informs the other sites of the location update (step ❸). This location update event includes mapping the user-device pair to both its new IP address as well as its new site. We discuss roaming in greater detail in Section 4.4.

Site-Specific Policy Slices: Network administrators commonly consider security policies to be confidential, as they provide valuable intelligence to attackers. A compromise of any individual

site should not reveal the global policy. To ensure this, MSNetViews provides per-site policies using a novel policy slicing technique that creates a "need-to-know" policy for each site.

Policy slicing mainly provides two benefits, (1) the attacker only learns the access control policy for the objects (network resources) of the compromised site and (2) it only learns about the users who have access to the compromised site's objects. The reduction in information about users is highly dependent on the type of resources and the policy definition itself. For example, if a site (e.g., the central bank branch) has resources (e.g., an email server) that nearly all employees should access, there will be a minimal reduction in the number of users.

Policy slicing works by identifying resource objects and users of a given site. Our slicing algorithm then traverses the policy graph building marking the subgraph of components connecting the users and resources. That subgraph defines the policy slice that each site will have access to. We discuss policy slicing formally in Section 4.5.

Policy Administration: Not all administrators can change the entire policy. MSNetViews uses *administrative policies* to limit what each administrator can do. MSNetViews also prevents policy errors using a policy checker that traverses the set of the candidate policies and evaluates then against a policy invariants. Violation of these invariants could lead to conflicts between the multiple policy classes. We discuss policy administration in Section 4.6.

4.4 Supporting Roaming Users

Geographically distributed enterprises have multiple distinct sites. Business operations frequently allow (or require) users to access local network resources *while physically present at a site*. For example, a policy may permit all users at site S_x to connect to printers and network-connected displays within S_x. Alternatively, a policy may restrict server access to only users at site S_x. This section considers a policy semantics and enforcement system with the flexibility to specify policies for users that move (roam) between multiple sites.[1]

4.4.1 Policy Support for Roaming

Handling roaming users presents two major policy-related challenges. First, the system must update policies, in real-time, in accordance with changing user locations. Second, those updated policies, and the enforcement of those policies, must be kept consistent across *all* sites in the enterprise. MSNetViews achieves dynamic policy updates using NGAC's concept of *obligations*, which are rules that accept events as input. Based on that input, the policy engine executes a set of pre-defined actions that change the set of enforceable rules. As a result, the administrator-defined, global policy is unchanged, thereby reducing the complexity of maintaining policy

[1]While we consider location on the granularity of a network site, our policy model is flexible enough to offer finer granularity (e.g., building or room), assuming the network enforcement mechanisms support it (e.g., RoArray [60]).

consistency across sites.

Modeling Location Policy

Shown in Figure 4.3, MSNetViews models location-specific policy by creating a separate policy class with separate user and object attributes. This separation (a) eases policy management by clarifying the two types of policies and (b) ensures the policy semantics remains intact. To demonstrate the importance for policy semantics, consider the case where both policy classes use object attribute oa, that is, $oa \rightsquigarrow pc_{role}$ and $oa \rightsquigarrow pc_{location}$. Any user that satisfies the location requirement ($u \rightsquigarrow ua$, $ua \rightsquigarrow pc_{location}$, and (ua, R, oa)) can then access any object o, when $o \rightsquigarrow oa$, regardless of the "Role" policy. Section 4.6.2 describes a policy checker to ensure administrators do not inadvertently violate this policy invariant.

The location policy class adheres to a general structure with two requirements: (1) there must be at least one user attribute per site, and (2) there must be a path between each object and at least one object attribute.

Administrators define *associations* between each user (or user attribute) and policy-relevant objects (or object attributes); these constitute the paths needed to satisfy the policy structural requirements. Thus, administrators can define the location requirements for each object by simply creating assignments from objects to the sites from which they should be accessible.

We include the user attribute, E (everyone), and the object attribute, RG (global resource), to simplify policy assignment of globally accessible resources (see Figure 4.3).

Dynamic Policy Update

Location updates trigger NGAC obligations. The MSNetViews policy-specification uses NGAC obligations to manage assignments between user-device pairs (NGAC users) and user attributes for the "Location" policy class. As such, the administrator-defined policy does not include explicit assignments from users to sites. For example, the dashed assignment from $\langle Alice, L1 \rangle$ to S_2 in Figure 4.3 is dynamically added. By omitting this assignment from the policy specification, MSNetViews needs only reestablish policy consistency across sites when a policy administrator updates the policy. Such policy actions are less frequent than network-join events of users, which can be as frequent as tens of users per second in large organizations.

At each site, external security events are passed to that site's NGAC event processing point (EPP), which causes the policy to execute the actions defined by matching rules. Obligations take the form $\langle ep, r \rangle$ where ep is an event pattern and r is a response. The response, r, is a set of actions. This pair is commonly read as **when** ep **do** r. NGAC obligations support both internal and external events, where internal events are triggered by policy actions (e.g., a policy query allows Alice to read file f) and external events are reported to the policy engine via an outward facing API.

An event pattern ep is a tuple $\langle u, pc, op, pe \rangle$, where a user $u \in U$ is performing an operation $op \in OP$ on a policy element $pe \in PE$ in policy class $pc \in PC$. We model location updates as external events. Each event describes an administrative ($u =$ `admin`) assignment ($op =$ `assign to`) of user-device pairs to the user attribute in the location policy class ($pc =$ `Location`) corresponding to the new site's location (e.g., $pe = S_2$). Upon receipt of a matching event pattern, the policy engine updates the location attribute assignments as appropriate. An environment with n sites requires n obligation rules, which are programmatically generated. The same obligation rules are used at all sites.

The following example obligation modifies assignments when a user-device pair updates its location to site S_2.

when $\langle$`admin`, `Location`, `assign to`, $S_2\rangle$
do `deassign`(`child_of_assign`, S_1)
`assign`(`child_of_assign`, S_2)
`deassign`(`child_of_assign`, S_3)
...
`deassign`(`child_of_assign`, S_n)

Informally, when an administrative event assigns a user-device pair to attribute S_2 in the `Location` policy class, then the policy deassigns that user-device pair from attributes $S_1, S_3, \ldots, S_n$. `child_of_assign` is a built-in function that returns the user-device pair that was assigned to S_2. We found that external events did not actually create the assignment being matched, therefore we include the assignment in the set of response actions.

4.4.2 System Support for Roaming

Shown in Figure 3.2, a user's physical movement between sites initiates a location and identity update. The local manager at the new site updates its policy state using the NGAC obligations described in Section 4.4.1. The local manager then informs the global manager, which in turn informs the local managers in the other sites (on a "need-to-know" basis, as discussed in Section 4.5). Those local managers update their policy state using their NGAC obligations. This design ensures that only the global manager is allowed to initiate updates to the policy state, identity, and location repositories. As such, MSNetViews avoids potentially malicious or unwanted local alterations. It also simplifies maintaining state consistency as the global manager serves as a permanent, central leader.

4.4.3 Policy State Consistency Across Sites

Whenever a user roams from one site to another, a location update event is triggered by the local manager of the new site. There is a short post-roaming delay between when the triggering site begins enforcing its new policy rules and when relevant remote sites begin enforcing their

new policy rules. This transmission delay is inherent in any distributed system. Fortunately, the resulting policy state inconsistency has limited impact in practice.

Policies can be inconsistent in two generalized scenarios. In the first scenario, a user may be unable to access a resource they should now have access to. As this inconsistency occurs only in the short period after the user attaches to a new network, the impact will be minimal. The local policy engine will shortly catch up with the pending updates, and in worst case, the user will only need to reattempt the connection (e.g., refresh a page in their web browser). In the second scenario, an inconsistency may theoretically allow a user to connect to a network resource they should no longer have access to. In practice, this is not a problem. When a user joins a new site, they will be assigned a different IP address than when in the previous site. Connections from the new IP address will fail. Attempts to forge the old IP address will also fail. If the destination is in the new site, the new site has the updated policy state. If the destination is not in the new site, the packet will not be routed out of the new site. We measure post-roaming stabilization in Section 4.8.

4.5 Policy Slicing

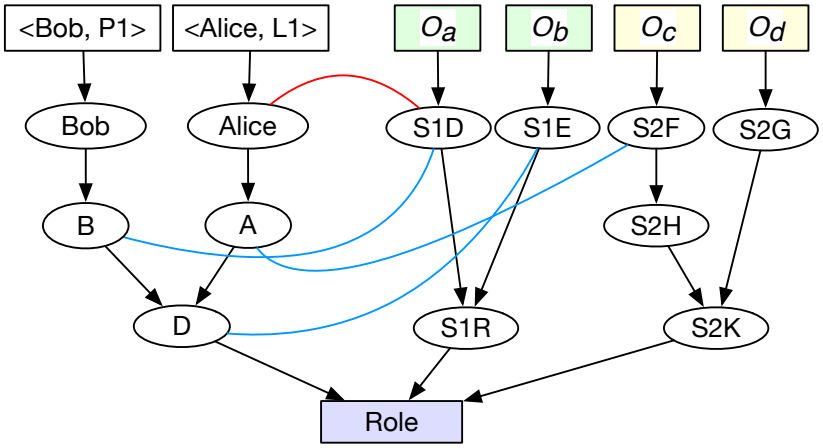
Organizations commonly consider their firewall policies to be confidential [96]. Intuitively, if an attacker has full knowledge of the firewall configuration, they can identify and more easily maneuver towards valuable targets. We mitigate policy exposure by proposing *policy slicing*, a method for distributing policy information based on the need of the site. Therefore, if a single site is compromised, the attacker cannot learn the global policy. It also forces attackers to perform active reconnaissance, which raises the probability of detection. This section presents our policy slicing algorithm and with an explanation of how MSNetViews selectively updates the policy slices for individual sites.

4.5.1 Policy Slicing Overview

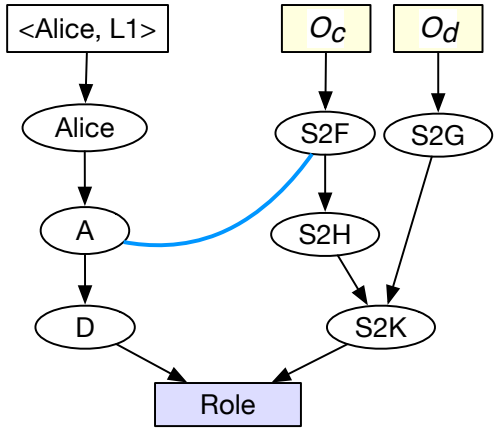
Each site only needs to know how to control access to *local resources*. As such, MSNetViews uses policy slicing to compute and distribute a *site-specific policy* to each site.

Definition 1 (Site-Specific Policy)**.** Given an MSNetViews policy $\mathcal{P}$ and a site S_x, a policy $\mathcal{P}_x$ is a site-specific policy if $\mathcal{P}_x \sqsubseteq \mathcal{P}$ and $\mathcal{P}_x$ only contains the policy elements, assignments, associations, and prohibitions needed to evaluate access control decisions for site S_x. $\sqsubseteq$ denotes the policy elements, assignments, associations, and prohibitions of $\mathcal{P}_x$ are a subset of those in $\mathcal{P}$.

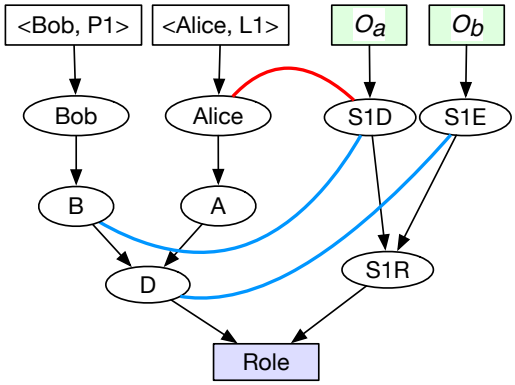
We note that in practice the policy slices created by our slicing algorithm in Section 4.5.2 are technically *near* least-information. In some cases, the algorithm leaves unnecessary user or object attributes that could be collapsed into a single new attribute. However, the benefit



(a) Global policy containing sites S_1 and S_2



(b) Policy slice for site S_2



(c) Policy slice for site S_1

Figure 4.4: Policy slicing example for the policy in Figure 4.3 (excluding location policy class for simplicity).

of collapsing unnecessary attributes is minimal, as the names of attributes can be obfuscated without loss of functionality. In contrast, not collapsing the unnecessary attributes simplifies the algorithm and proofs of correctness (i.e., the slice is correct by construction).

Figure 4.4 provides an example (excluding location policy class from Figure 4.3) that depicts the high-level intuition behind policy slicing. Intuitively, the policy slices for site S_1 (Fig. 4.4b) and site S_2 (Fig. 4.4c) are created by identifying the objects specific to each site and then finding all of the object attributes, user attributes, and users that refer to those objects by inspecting the assignments, associations, and prohibitions.

As shown in Figure 4.4b, site S_2 only needs the subset of the policy relevant to resource O_c and O_d. This site-specific policy includes object attributes $S2G$, $S2F$, $S2H$, and $S2K$ are also included for directly or indirectly assigned to resource O_c and O_d. Furthermore, the policy includes user attribute A, because there is an association rule connecting A to $S2F$. Next, the policy includes user $\langle Alice, L1\rangle$, because $\langle Alice, L1\rangle \rightsquigarrow A$. The policy slice for site S_1 in Figure 4.4c is constructed similarly and also considers prohibition relations.

Note that additional policy protection can be achieved by obfuscating the names of user and object attributes. If an administrator wishes to have useful information for debugging, only attributes *without* associations or prohibitions should be obfuscated.

Note that the slice for site S_2 includes user attributes *Alice* and D and object attributes $S2G$, $S2H$, and $S2K$. As discussed above, these attributes are not technically necessary, but their inclusion simplifies slice creation and correctness. If these attribute names are obfuscated, an attacker learns negligible information about the global policy outside of what is necessary to enforce the policy in the local site.

Policy Slicing Security Benefit: Policy slicing provides three types of benefits if an attacker access the policy at a given site S_x. First, the attacker only learns the access control policy for the objects (network resources) in site S_x and not the rest of the organization. Second, the attacker only learns about the users who have access to the objects in site S_x. The reduction in number of users is highly dependent on the policy. If the network resources in a given site are only accessed by a relatively small number of users, policy slicing will have very large benefit. On the other hand, if a site has resources (e.g., email server) that should be accessed by nearly all employees, there will be a very small benefit. Finally, the attacker only learns about the organizational roles (e.g., human resources, IT) corresponding to user attributes in the tree that has associations or prohibitions to network resources in site S_x. As noted above, the names of all user and object attributes can be obfuscated without impacting policy evaluation.

4.5.2 Policy Slicing Algorithm

The policy slicing algorithm takes as input (1) a global policy, (2) a site S_x, and (3) the set of objects for S_x. It outputs a site-specific policy (Def. 1). The algorithm traverses the global

policy to identify all policy elements, assignments, associations, and prohibitions to determine which ones are *relevant* for site S_x. Our algorithm is based on a series of definitions of relevant elements (denoted as subscript x). The relevant elements build towards the final algorithm.

Definition 2 (Relevant Objects)**.** Let O be the set of objects in the global policy $\mathcal{P}$. The set of relevant objects $O_x \subseteq O$ for site S_x is the set of network resources that reside in site S_x.

The set of *relevant object attributes* are all the object attributes on a path between a relevant object and a policy class. This set is equivalent to all the object attributes with a path to a relevant object. All objects are also object attributes.

Definition 3 (Relevant Object Attributes)**.** Let OA be the set of object attributes in the global policy $\mathcal{P}$ and O_x be the set of relevant objects (Def. 2). The set of relevant objects OA_x is the set $\{oa \mid oa \in OA \wedge \exists o \in O_x, o \rightsquigarrow oa\} \cup O_x$.

The *relevant associations* and *relevant prohibitions* are those that target relevant object attributes.

Definition 4 (Relevant Associations)**.** Let $\mathcal{R}_a$ be the set of associations in the global policy $\mathcal{P}$ and OA_x be the set of relevant object attributes (Def. 3). The set of relevant associations $\mathcal{R}_{a,x}$ is the set $\{r \mid r \in \mathcal{R}_a \wedge r = (ua, \_, oa) \wedge oa \in OA_x\}$.

Definition 5 (Relevant Prohibitions)**.** Let $\mathcal{R}_p$ be the set of associations in the global policy $\mathcal{P}$ and OA_x be the set of relevant object attributes (Def. 3). The set of relevant prohibitions $\mathcal{R}_{p,x}$ is the set $\{r \mid r \in \mathcal{R}_p \wedge r = (ua, \_, oa) \wedge oa \in OA_x\}$.

Relevant user attributes require careful consideration. This set includes both the user attributes referenced by a relevant association *and* all of the user attributes on paths between users and policy classes, but only those which traverse through user attributes referenced by a relevant association or prohibition.

Definition 6 (Relevant User Attributes)**.** Let UA be the set of user attributes in the global policy $\mathcal{P}$, $\mathcal{R}_{a,x}$ be the relevant associations (Def. 4), and $\mathcal{R}_{p,x}$ be the relevant prohibitions (Def. 5). Let UA'_x be the set $\{ua \mid ua \in UA \wedge [\exists(ua, \_, oa) \in \mathcal{R}_{a,x} \vee \exists(ua, \_, oa) \in \mathcal{R}_{p,x}]\}$. The set of relevant user attributes UA_x is $\{ua \mid ua \in UA \wedge \exists ua' \in UA'_x, (ua \rightsquigarrow ua' \vee ua' \rightsquigarrow ua)\} \cup UA'_x$.

The set of *relevant users* is then straightforward to define. Note that unlike objects, users are not user attributes.

Definition 7 (Relevant Users)**.** Let U be the set of users in the global policy $\mathcal{P}$ and UA_x be the set of relevant user attributes (Def. 6). The set of relevant users U_x is $\{u \mid u \in U \wedge \exists ua \in UA_x, u \rightsquigarrow ua\}$.

Algorithm 1 Create Local Policy For Site S_x

```
1: procedure CreateLocalPolicy(P, O_x)
2:     E = GetRelevantElements(P, O_x)
3:     R = GetRelevantRelations(P, E)
4:     return P_x = E ∪ R
5: procedure GetRelevantElements(P, O_x)
6:     OA_x = O_x, U_x = ∅, UA_x = ∅, R_{a,x} = ∅, R_{p,x} = ∅
7:     for each o ∈ O_x
8:         OA'_x = getAssignedOAs(P, o)
9:         R_{a,x} = R_{a,x} ∪ getAssociations(P, OA'_x)
10:        OA_x = OA_x ∪ OA'_x
11:    for each (ua, _, oa) ∈ {R_{a,x} ∪ R_{p,x}}
12:        UA_p = getParentUAs(P, ua)
13:        UA_c = getChildUAs(P, ua)
14:        U_x = U_x ∪ getUsers(P, ua)
15:        UA_x = UA_x ∪ ua ∪ UA_p ∪ UA_c
16:    return E = U_x ∪ UA_x ∪ OA_x ∪ getPCs(P)
17: procedure GetRelevantRelations(P, E)
18:    R_x = ∅
19:    for each e ∈ E
20:        for each r ∈ getAssignments(P, e)
21:            if r.source ∈ E and r.target ∈ E then
22:                R_x = R_x ∪ r
23:        for each r ∈ getAssociations(P, e)
24:            if r.source ∈ E and r.target ∈ E then
25:                R_x = R_x ∪ r
26:        for each r ∈ getProhibitions(P, e)
27:            if r.subject ∈ E and r.container ∈ E then
28:                R_x = R_x ∪ r
29:    return R_x
```

The final information required to create the site-specific policy is the set of *relevant assignments*, which are all of the assignments related to the relevant policy elements.

Definition 8 (Relevant Assignments)**.** Let $\mathcal{R}_e$ be the set of assignments in the global policy and $PE_x = U_x \cup UA_x \cup OA_x \cup PC$ be the set of relevant policy elements (Defs. 3, 6, 7). The set of relevant assignments $\mathcal{R}_{e,x}$ is $\{r \mid r \in \mathcal{R}_e \wedge \exists e_1, e_2 \in PE_x, r = (e_1, e_2)\}$.

Our policy slicing algorithm produces a "near" (see Section 4.5.1) least-information policy by calculating the relevant object attributes, user attributes, users, assignments, associations, and prohibitions based on the above definitions.

Algorithm 1 summarizes the entire process, which follows directly from the definitions. Finally, we do not discuss *relevant obligations*, as we limit our use of obligations to the creation

of assignments for managing user roaming between sites (Section 4.4). We leave the treatment of additional obligation types to future work.

4.5.3 Managing Policy Updates

Network administrators will update the global policy over time. However, not every update will impact all sites. MSNetViews optimizes policy updates by only generating and distributing policy slices for sites impacted by the policy update. Our key intuition is that the set of impacted sites can be easily determined by considering the set of policy elements that are impacted by the policy change. If a site does not contain any impacted policy elements, its policy slice does not need to be regenerated and retransmitted.

Definition 9 (Impacted Policy Elements)**.** Let $\mathcal{P}$ be a global policy and $\mathcal{P}'$ be an update to $\mathcal{P}$. The set of impacted policy elements is determined by the function $\delta(\mathcal{P}, \mathcal{P}')$, which includes: (1) all policy elements added or removed and (2) all policy elements referenced by an assignment, association, or prohibition that has been added, removed, or changed.

In Section 4.7, we prove that if the policy $\mathcal{P}_x$ for site S_x does not contain an impacted policy element, then the policy slice for site S_x for $\mathcal{P}$ is identical to its the policy slice for $\mathcal{P}'$.

4.5.4 Cross-Site Traffic

MSNetViews's site-specific policies ensure that each site only has enough information to enforce access on *local resources*. While this goal minimizes policy disclosure if a site is compromised, it requires an allow-all-outbound policy between sites. However, if not carefully addressed, hosts are vulnerable to unauthorized, cross-site accesses by abusing this allow-all-outbound policy to circumvent the global policy. Furthermore, MSNetViews builds on NetViews [9], which allows reply traffic to *any* client source port as a performance optimization.

Figure 4.5 depicts a simplified attack scenario where the attacker host h_a is in site S_2 and the victim host h_v is in site S_1. **Setup:** The attacker tricks h_v into making a connection to h_a (e.g., via a resource on a web page). The first packet is allowed to exit S_1 via gateway G_1 but is denied at sw_2 when it enters S_2. At this point, S_1 is configured to allow reply traffic from host h_a. **Attack:** Host h_a now connects to h_v. The allow-all-outbound policy in site S_2 directs S_2's gateway G_2 to forward the packet to site S_1. Switch sw_1 then delivers the packet to h_v, because it appears to be valid return traffic from the original connection.

Note that h_a can connect to *any* port on host h_v, because NetViews [9] optimizes for applications that commonly use multiple connection requests (e.g., HTTP) by allowing any client source port, which avoids redundant flow rules. This attack is unique to the multisite setting with a least-information policy. This attack is not possible in a single-site setting, because the connection to h_a would have been denied and the reply path would not have been set up.

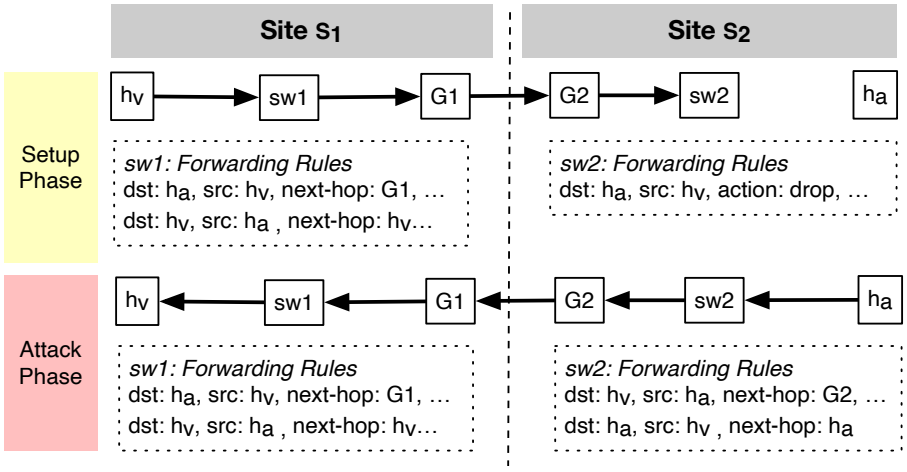


Figure 4.5: Policy slicing introduces an attack scenario for cross-site traffic. Since sites S_1 and S_2 have incomplete information, and they must allow all outbound traffic.

Similarly, without a site-specific policy in multisite setting, the global policy would have enforced the policy at the source-site.

The attack is a result of *unverified reverse cross-site traffic*. If site S_2 knows that site S_1 denied the packet, it will not confuse h_a's connection attempt with valid reply traffic. This insight leads us to two possible solutions. The simplest solution is for the destination site to securely inform the originating site that the packet was denied as it entered the site. It will allow the originating site to remove the corresponding forwarding rules. In this case, the NetViews instance running in the originating site can remove the SDN forwarding rules before the allowed reply-flow is abused. A more elegant solution is for destination sites to include an unforgeable *validation bit* for all reply traffic allowed by the local policy. The gateway could ensure that a designated bit in the IP header is only set for reply traffic. The originating site then will only allow reply-traffic from the trusted destination site if the validation is bit set. The validation bit solution can be efficiently implemented in reactive SDN environments that allow manipulation of headers in addition to forwarding. For example, the ONOS Intents framework (used by NetViews to allow reply traffic) can also be used to define `FlowMod` rules that (a) set the validation bit in the destination site and (b) require the validation bit in the originating site. With this solution in place, the attack packet from h_a is not a reply of an allowed packet and hence the validation bit will not be set. Since the attack packet is not expected, it will cause a `PacketIn` when entering site S_2, where it will be denied. Note that each site should clear the validation bit for all non-reply traffic to prevent forgery.

4.6 Administration of Policy

Enterprises often have many policy administrators with at least one administrator per site. MSNetViews requires all policy changes to occur within the centralized global manager. We assume a secure admin-console that allows administrators to use their private credentials to access the global policy. This section describes how MSNetViews (1) limits the privileges of individual policy administrators and (2) prevents policy administrators from making errors that unintentionally change the policy semantics.

4.6.1 Administrative Policy

MSNetViews uses NGAC's existing administrative policy model for specifying what actions individual policy administrators can perform. The left side of Figure 4.6 depicts an example NGAC administrative policy that defines administrative relations to the non-administrative policy.

Administrative Authority: NGAC administrative policies organize administrators into users and user attributes just as with non-administrative policy. By convention, NGAC defines three levels of administrative authority: (1) *Principal Authority* (PA), (2) *Domain Administrator* (DA), and (3) *Subdomain Administrator* (SA). DAs and SAs are subordinate to PAs. A PA is authorized to access the entire global policy. It is the only authority with the ability to create a policy class. PAs can also create one or more *segments* in the global policy. This allows a portion of policy to be declared and handled separately. The PA can then create and assign subordinate administrators to manage different segments of the global policy. DAs can access users, objects, and attributes within its assigned domain and manage the entire domain, if permitted by the PA. In an enterprise setting, a DA is the appropriate authority for a site administrator, as it manages the policy elements relevant to its own site (as shown in Figure 4.6). A DA can also define a subdomain of its domain and allocate select privileges to an SA.

Administrative Relations: An administrator's ability to update the enterprise policy (i.e., the non-administrative policy) is determined by *administrative relations*, which are analogous to the non-administrative policy relations described in Section 2.5. Administrative *associations* grant administrative access rights including: (1) addition or deletion of a policy element, (2) addition or deletion of assignments between policy elements, and (3) addition, deletion or update of an association, prohibition, or obligation. Administrative *prohibitions* and *obligations* are the same as their non-administrative counterparts, only with administrative operations.

4.6.2 Policy Checker

MSNetViews ensures each update from a policy administrator is valid and does not contain errors that unintentionally change the policy semantics. Policy checks are needed for (a) *updating*

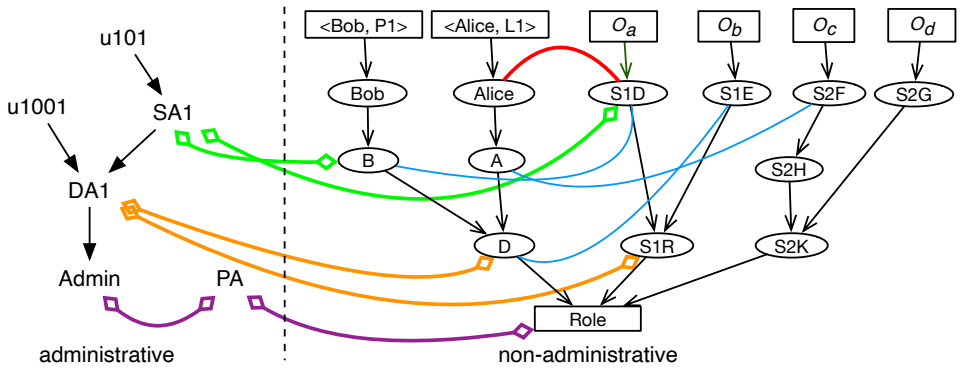


Figure 4.6: Example administrative policy. The left side defines an administrative policy. The ◇-ended lines show administrative access rights on non-administrative policy (excluded location policy class from Figure 4.3 for simplicity).

Table 4.1: MSNetViews's Additional Policy Check

Rule	**Purpose**
(1) Dangling PE	Each $pe \in PE$ must lead to at least one $pc \in PC$
(2) Exclusive UA	Each $ua \in UA$ must lead to only one $pc \in PC$
(3) Exclusive OA	Each $oa \in OA$ must lead to only one $pc \in PC$
(4) Exclusive Associations	The source and target attributes of an association relation must lead to same policy class.
(5) Exclusive Prohibitions	The source and target attributes of a prohibition relation must lead to same policy class.

policy elements (users, objects, object-attributes, or user-attributes) and (b) *updating relations* (assignments, associations, or prohibitions). The MSNetViews uses NGAC policy engine reference implementation (policy-machine-core) [75], which includes checks for all the NGAC-defined syntax and graph-related inconsistencies (e.g., existence of loops). However, we found several limitations with NGAC's existing policy checks. We also identified inconsistencies between the NGAC policy definition [45] and the implemented checks.

Missing Checks for Policy Elements: The NGAC documentation [45] states that a policy element should not be declared without a corresponding assignment relation. Such *dangling policy elements* introduce inconsistencies in policy enforcement and mediation of the location-context. We found that the policy-machine-core implementation does not perform this check.

Missing Checks for Policy Relations: Adding a policy element therefore also requires adding an assignment relation. The policy-machine-core implementation ensures NGAC's requirement that assignment relations are acyclic, irreflexive, and not defined from an object attribute to an object. However, neither the NGAC documentation or implementation ensure that policy elements have a path to only one policy class, which is important for ensuring *separation between the policy classes*. As discussed in Section 4.4, sharing user or object attributes

between policy classes can unintentionally override rules.

Next, the policy-machine-core performs syntax, redundancy, existence, and similar checks for associations and prohibitions in the NGAC documentation. However, we observe that associations and prohibitions referencing policy elements belonging to multiple policy classes contradict MSNetViews' location-aware policy enforcement. Therefore, additional checks are required.

Policy Check Enhancements: Table 4.1 lists the five policy checks we added the policy-machine-core. Rule 1 ensures the policy does not include any dangling policy elements that do not lead to a policy class.

Rule 1 (Dangling PE)**.** Let PE be the policy elements, $PE \stackrel{def}{=} U \cup UA \cup OA \cup PC$. Where U is the set of users, UA is the set of user attributes, $OA \supseteq O$ is the set of object attributes (includes objects), and PC is the set of policy classes. If x is a policy element (i.e., $x \in (PE \setminus PC)$), then it must have a path of assignment relations towards at-least one policy class node $pc \in PC$, $x \rightsquigarrow pc$.

Rules 2 and 3 ensure that user and object attributes can only lead to one policy class.

Rule 2 (Exclusive UA)**.** A user attribute cannot belong to multiple policy classes. Let, UA is the set of user attributes and PC is the set of policy classes. If $x, y \in UA$ and $p, q \in PC$, then: $(x \rightsquigarrow p, y \rightsquigarrow q) \Rightarrow x \neq y$.

Rule 3 (Exclusive OA)**.** A object attribute cannot belong to multiple policy classes. Let, OA is the set of object attributes, O is the set of objects and PC is the set of policy classes. If $x, y \in (OA \setminus O)$ and $p, q \in PC$, then: $(x \rightsquigarrow p, y \rightsquigarrow q) \Rightarrow x \neq y$.

Finally, Rules 4 and 5 ensure that associations and prohibitions do not combine attributes from two different policy classes, which ensures the separation of the two policies.

Rule 4 (Exclusive Associations)**.** The association relation should be between nodes that belong to the same policy class. Let UA be the set of user attributes, OA the set of object attributes, O the set of objects, PC the set of policy classes, and $\mathcal{R}_a$ the set of association relations, respectively. If $x \in UA$, $y \in (OA \setminus O)$ and $p, q \in PC$, then: $(x \rightsquigarrow p, y \rightsquigarrow q) \Rightarrow (x, ars, y) \notin \mathcal{R}_a$.

Rule 5 (Exclusive Prohibitions)**.** The prohibition relation should be between nodes that belong to the same policy class. Let UA be the set of user attributes, OA the set of object attributes, O the set of objects , PC the set of policy classes, and $\mathcal{R}_p$ the set of prohibition relations, respectively. If $x \in UA$, $y \in (OA \setminus O)$ and $p, q \in PC$, then: $(x \rightsquigarrow p, y \rightsquigarrow q) \Rightarrow (x, ars, y) \notin \mathcal{R}_p$.

Implementation: MSNetViews policy checker is built upon the NGAC policy engine implementation (policy-machine-core) [75]. A new policy or updated policy goes as input to the policy checker, the main policy file (which has the encoding of the policy graph) is checked first. All

Table 4.2: The Consolidated List of Policy Machine Checks

Rule	Purpose
(1)	All nodes need to be unique
(2)	Node type can be only $\langle OA, UA, U, O, PC, OS \rangle$
(3)	Node declaration need at least three parameter. Node can can not have a null or empty name, type and property
(4)	Assignment relation need child node and parent node only
(5)	Node with type PC can not be assigned to any thing
(6)	Node with type OA can only be assigned to node with type PC and OA
(7)	Node with type O can only be assigned to node with type OA
(8)	Node with type UA can only be assigned to node with type PC and UA
(9)	Node with type U can only be assigned to node with type UA
(10)	There can not multiple assignments between a pair of nodes
(11)	Association relation need at least three parameter
(12)	Association relation from UA to $\{UA, OA\}$ are only allowed
(13)	If there is already an association between two nodes, the current operation set (swap, not add) will be updated with new one
(14)	Prohibition can not be null, name can not be null, subject can not be null
(15)	There is checks provided for necessary fields on Obligation structure in general, event pattern, actions and functions

the NGAC-defined syntaxes and graph-related inconsistencies (e.g., the existence of a loop) for the main policy are checked in the policy-machine-core's `GraphSerializer` (Table 4.2). We added tests for Rules (1)-(4). NGAC uses a separate prohibition input files, which are checked in the policy-machine-core's `ProhibitionsSerializer`, which we extended to enforce Rule (5). For obligations, we implemented the specific syntaxes needed for location change events (defined in Section 4.4.1).

4.7 Security Analysis

MSNetViews transforms the role-based NetViews policy in four ways that impact the NGAC policy enforced by SDN in the individual sites. First, MSNetViews creates policy slices, which are a subset of the global policy. Each site has its own policy slice. Second, when the global policy is updated, MSNetViews only generates a new policy slice for a site if it things the update impacts the policy slice for that site. If this update detection logic is incorrect, then the site may enforce an outdated version of the policy. Third, MSNetViews adds a second policy class for incorporating location. If not defined with care, policy classes can interfere with one another. Fourth and finally, MSNetViews uses obligations to dynamically update the policy based on

user location changes. This section demonstrates the security of MSNetViews's enforcement by proving the following theorem.

Theorem (MSNetViews Enforcement)**.** *The policy MSNetViews enforces at each site is at least as restrictive as the role-based policy class in the global policy.*

Before proving this theorem, we present three lemmas that prove specific properties about policy slicing, policy updating, and multiple policy classes. After presenting these lemmas, we return to the proof of the theorem

4.7.1 Impact of Policy Slicing

Network administrators commonly consider security policies to be confidential, as they provide valuable intelligence to attackers. Hence, a compromise of any individual site should not reveal the global policy. MSNetViews maintains this policy confidentiality through providing per-site policies using a novel policy slicing technique. It creates a "need-to-know" policy for each site inspired by "program slicing." The following lemma proves that policy slices match the global policy for all objects (hosts) in a site.

Lemma 1 (Policy Slice Correctness)**.** Let $\mathcal{P}$ be a global policy and $\mathcal{P}_x$ be site S_x's policy slice for objects O_x. Policy slice $\mathcal{P}_x$ is *correct* if there does not exist a user u that has *more* or *less* access rights to an object $o \in O_x$ in $\mathcal{P}_x$ than in $\mathcal{P}$.

Proof. We begin by proving that a user u cannot have *more access rights* in $\mathcal{P}_x$. Access rights are granted by either (a) adding an association that grants access rights; (b) adding an assignment that creates an assignment path from user u to a user attribute that has an association granting access rights; (c) adding an assignment that creates an assignment path from an object o to an object attribute that is the target of an association inherited by user u; or (d) the removal of a prohibition that restricted access rights. Since the policy slicing algorithm does not add any assignments or associations, the first three cases do not apply. Based on the definition of relevant prohibitions (Def. 5), prohibitions are not removed if they reference a relevant object attribute (Def. 3), which are all of the object attributes that have a path from all objects in O_x. Therefore, no prohibitions related to objects in O_x are removed. Hence user u cannot have more access rights in $\mathcal{P}_x$.

Next we prove that a user u cannot have *less access rights* in $\mathcal{P}_x$. Access rights are removed by either (a) adding prohibitions; (b) removing an association that grants access rights; (c) removing an assignment on an assignment path from user u to a user attribute with an association granting access rights; (d) removing an assignment on an assignment path from an object in O_x to an association granting access rights; or (e) removing a user from the policy. Since the policy slicing algorithm does not add any prohibitions, case (a) does cannot occur. Based on the definition of relevant object attributes (Def. 3), case (d) cannot occur. Following this and

the definition of relevant associations (Def. 4), case (c) cannot occur. Following this and the definition of relevant user attributes (Def. 6), case (b) cannot occur. Finally, following this and the definition of relevant users (Def. 7), case (e) cannot occur. Therefore, user u cannot have less access rights in $\mathcal{P}_x$. □

4.7.2 Impact of Optimized Distribution for Policy Updates

Network administrators will update the global policy over time. However, not every update will impact all sites. MSNetViews optimizes policy updates by only generating and distributing policy slices for sites impacted by the policy update. Our key intuition is that the set of impacted sites (Definition 9) can be easily determined by considering the set of policy elements that are impacted by the policy change. If a site does not contain any impacted policy elements, its policy slice does not need to be regenerated and retransmitted.

We now prove that if the policy $\mathcal{P}_x$ for site S_x does not contain an impacted policy element, then the policy slice for site S_x for $\mathcal{P}$ is identical to its the policy slice for $\mathcal{P}'$.

Lemma 2 (Identical Policy Slice)**.** Let $\mathcal{P}$ be a global policy and $\mathcal{P}'$ be an update to $\mathcal{P}$. Let $\mathcal{P}_x$ and $\mathcal{P}'_x$ be site S_x's policy slices for $\mathcal{P}$ and $\mathcal{P}'$, respectively. Let PE_x be the set of policy elements in $\mathcal{P}_x$. If $\delta(\mathcal{P}, \mathcal{P}') \cap PE_x = \emptyset$, then $\mathcal{P}_x = \mathcal{P}'_x$.

Proof. We provide a proof by contradiction. Assume $\mathcal{P}_x \neq \mathcal{P}'_x$. Let PE_x and PE'_x be the policy elements in $\mathcal{P}_x$ and $\mathcal{P}'_x$, respectively. Recall $\mathcal{P}_x \sqsubseteq \mathcal{P}$ and $\mathcal{P}'_x \sqsubseteq \mathcal{P}'$ by definition. The difference that causes $\mathcal{P}_x \neq \mathcal{P}'_x$ could only result from the following scenarios: (a) adding or removing a policy element, (b) adding or removing an assignment, association, or prohibition, or (c) changing the policy elements referenced by an assignment, association, or prohibition.

If policy element e_1 was added ($e_1 \in PE'_x$ and $e_1 \notin PE_x$), then there must exist an assignment in $\mathcal{P}'_x$ from e_1 to a policy element e_2 where e_2 is in PE_x. This is because all users, objects, and attributes have paths that end in a policy class, and the set of policy classes is fixed. Since e_1 is new, the assignment did not exist in $\mathcal{P}_x$. Therefore $e_2 \in \delta(\mathcal{P}, \mathcal{P}')$. Since e_2 cannot be in both PE_x and $\delta(\mathcal{P}, \mathcal{P}')$, this is a contradiction.

If a policy element e_1 was removed ($e_1 \in PE_x$ and $e_1 \notin PE'_x$), either e_1 was removed from the global policy $\mathcal{P}$ or it was only removed from $\mathcal{P}_x$ but remains in the global policy. If e_1 was removed from the global policy, then e_1 must be in $\delta(\mathcal{P}, \mathcal{P}')$, which is a contradiction. If e_1 was not removed from the global policy, but it was removed from $\mathcal{P}_x$, then there must exist an assignment in $\mathcal{P}_x$ from e_1 to a policy element e_2 where e_2 is in PE_x. This situation can only occur if that assignment was removed, which would cause e_2 to be in $\delta(\mathcal{P}, \mathcal{P}')$. Since e_2 cannot be in both PE_x and $\delta(\mathcal{P}, \mathcal{P}')$, this is a contradiction.

The remaining cases are trivial. If no policy elements are added or removed, and there is an added, removed, or changed assignment, association, or prohibition, it must refer to a

policy element e in PE_x. However, e would then also be in $\delta(\mathcal{P}, \mathcal{P}')$, which is a contradiction. Therefore, $\mathcal{P}_x = \mathcal{P}'_x$. □

4.7.3 Impact of Multiple Policy Classes

MSNetViews uses multiple policy classes to encode the role and location policy rules. By definition, NGAC only allows access if access is allowed by all policy classes. However, the semantics of a policy may not be clear to a policy administrator if two policy classes interfere with one another. We define policy class interference as follows.

Definition 10 (Policy Class Interference)**.** Let $\mathcal{P}$ be an NGAC policy with two policy classes, pc_1 and pc_2. Let $\mathcal{P}_1 \sqsubseteq \mathcal{P}$ and $\mathcal{P}_2 \sqsubseteq \mathcal{P}$ be the policy subsets for policy classes pc_1 and pc_2, respectively. We say that $\mathcal{P}_1$ and $\mathcal{P}_2$ *interfere* in $\mathcal{P}$ if there exists a user-object pair (u, o) with access rights AR in $\mathcal{P}$, but (u, o) does not have access rights AR in *both* $\mathcal{P}_1$ and $\mathcal{P}_2$.

MSNetViews prevents policy class interference using Invariants 2-5. These invariants effectively isolate the role and location policy classes by preventing then from sharing any user attributes, object attributes, associations, or prohibitions. We prove that these invariants prevent policy class interference as follows.

Lemma 3 (Policy Class Isolation)**.** Let $\mathcal{P}$ be an NGAC policy with two policy classes, pc_1 and pc_2. Let $\mathcal{P}_1 \sqsubseteq \mathcal{P}$ and $\mathcal{P}_2 \sqsubseteq \mathcal{P}$ be the policy subsets for policy classes pc_1 and pc_2, respectively. If $\mathcal{P}_1$ and $\mathcal{P}_2$ do not share any user attributes, object attributes, associations, or prohibitions, then their policy classes do not interfere.

Proof. By definition, when a policy has more than one policy class, NGAC only allows user u to access object o with access rights AR if there exists an association relations granting AR for all policy classes. More specifically, for each policy class, there must exist association relation (ua, AR', oa), where $AR \subseteq AR'$, and assignment paths $u \rightsquigarrow ua \rightsquigarrow pc$, $o \rightsquigarrow oa \rightsquigarrow pc$. Note that, if *any* of the policy classes has a prohibition relation denying any rights in AR for user u and object o (defined similarly via assignment paths), then access is denied. However, since our definition of policy class interference (Def 10) does not consider a reduction of access rights, we do not need to consider prohibition relations in this proof.

We prove that policy class isolation prevents policy interference by contradiction. Assume there is policy interference, that is there exists a user u that can access object o (with any rights) in both $\mathcal{P}$ and $\mathcal{P}_1$, but not $\mathcal{P}_2$. For pc_2 to be satisfied, $\mathcal{P}$ must contain an association relation $(ua, \_, oa)$, where $u \rightsquigarrow ua \rightsquigarrow pc_2$, $o \rightsquigarrow oa \rightsquigarrow pc_2$.

Consider the case where $\mathcal{P}_2$ contains the association relation $(ua, \_, oa)$. Since $\mathcal{P}_1$ and $\mathcal{P}_2$ cannot share user or object attributes, ua and oa are in $\mathcal{P}_2$ and not $\mathcal{P}_1$. Since $\mathcal{P}_2$ does not allow u to access o, then at least one of the necessary paths does not exist: $u \not\rightsquigarrow ua$, $ua \not\rightsquigarrow pc_2$,

$o \not\rightsquigarrow oa$, or $oa \not\rightsquigarrow pc_2$. Furthermore, since $\mathcal{P}$ does have all of those paths, then the missing assignment relation must have been defined in $\mathcal{P}_1$. However, if this is the case, then $\mathcal{P}_1$ contains an assignment relation that references at least one user or object attribute in $\mathcal{P}_2$. Since this cannot occur, this case is a contradiction.

Next, consider the case where $\mathcal{P}_2$ does not contain the association relation $(ua, \_, oa)$. Therefore, the association relation exists in $\mathcal{P}_1$, which implies $\mathcal{P}_1$ also contains ua and oa. Since user and object attributes cannot be in both $\mathcal{P}_1$ and $\mathcal{P}_2$, ua and oa are not in $\mathcal{P}_2$. However, if $\mathcal{P}$ has paths $ua \rightsquigarrow pc_2$, and $oa \rightsquigarrow pc_2$, there must exist a user attribute ua' and object attribute oa' in $\mathcal{P}_2$ such that $ua \rightsquigarrow ua' \rightsquigarrow pc_2$, $oa \rightsquigarrow oa' \rightsquigarrow pc_2$. However, for this to occur, there must be assignments in $\mathcal{P}_1$ referencing ua' and oa', which is a contradiction. □

4.7.4 MSNetViews Enforcement Proof

We now return to the MSNetViews enforcement theorem, which states

> *The policy MSNetViews enforces at each site is at least as restrictive as the role-based policy class in the global policy.*

Proof. NetViews defines a single role-based policy class. The MSNetViews policy differs from the NetViews policy in four ways: (1) it creates a policy slice for each site, (2) it optimizes the distribution of policy updates by only generating a new policy slice if the previous policy slice references a changed policy element, (3) it includes an additional policy class for location, and (4) it defines obligations to dynamically update the policy when users change locations. None of these changes result in a policy decision that is less restrictive than the role-based policy class that defines a NetViews policy.

Let $\mathcal{P}$ be the global policy and $\mathcal{P}_x$ be the policy slice for site S_x. Based on Lemma 1 (Policy Slice Correctness), the policy for the role-based policy class in $\mathcal{P}_x$ is just as restrictive the role-based policy class in $\mathcal{P}$ for the objects O_x in site S_x.

When the global policy $\mathcal{P}$ is updated to $\mathcal{P}'$, MSNetViews only updates the policy at site S_x if $\mathcal{P}_x$ references one of the of policy elements impacted by the update. Based on Lemma 2 (Identical Policy Slice), if MSNetViews had re-generated policy slice for $\mathcal{P}'$, the resulting policy slice $\mathcal{P}'_x$ would have been identical to $\mathcal{P}_x$. Therefore, the optimized distribution for policy updates does not impact the restrictiveness of the role-based policy class.

Next, the addition of the location policy class results in a policy that is at least as restrictive as the role-based policy class. As proved by Lemma 3 (Policy Class Isolation), MSNetViews's policy invariants prevents the policy class interference between the location policy class and the role-based policy class. Note that our definition of policy class interference (Def 10) only considers the creation of additional access rights. The location can (and does) result in a policy that is more restrictive than the role-based policy class.

Table 4.3: Evaluation Topologies

Topology	Devices	Switches	Details
Cisco [155]	12	10	Network of an enterprise with Cisco PIX firewall
MiniStanford [155]	100	25	Stanford backbone network

Finally, the dynamic location updates does not impact the role-based policy class semantics. MSNetViews's NGAC obligations only modify assignments to user attributes used by the location policy class. Since the location policy class is isolated from the role-based policy class, and Lemma 3 (Policy Class Isolation) proves the location policy class cannot make the policy less restrictive than the role-based policy class, the obligations also cannot make the policy less restrictive. □

4.8 Performance Evaluation

In this section, we evaluate the performance of MSNetViews through the following research questions:

Q1 How does MSNetViews enforcement impact system latency and throughput? (Section 4.8.2)

Q2 How long does MSNetViews take to stabilize after a user roams between sites? (Section 4.8.3)

Q3 How expensive are policy checking and update operations, and do they scale? (Section 4.8.4)

4.8.1 Network Emulation Methodology

Emulated Network Design: We emulate our enterprise networks using Mininet [140] and ONOS [56]. Because no enterprise has publicly released its network topology, we model a multisite enterprise as two sites with identical topologies. Performing the performance evaluation with only two sites is sufficient, as it represents the case where all sites have direct WAN connections to one another. We emulate this site-to-site connection with a low-overhead GRE tunnel over a 10GbE WAN link. We vary the WAN latency to emulate connections between sites in the same city (`WashingtonDC`↔`WashingtonDC`, 1 ms), same region (`WashingtonDC`↔`NY`, 11.2 ms), and across the Atlantic (`WashingtonDC`↔`Copenhagen(CP)`, 105 ms). These latency values come from WonderNetwork's global `ping` stats [150], and the specific cities were chosen so that the latencies were roughly round numbers an order of magnitude apart. Within sites, we use the same topologies used in prior work [9], shown in Figure 3.7.

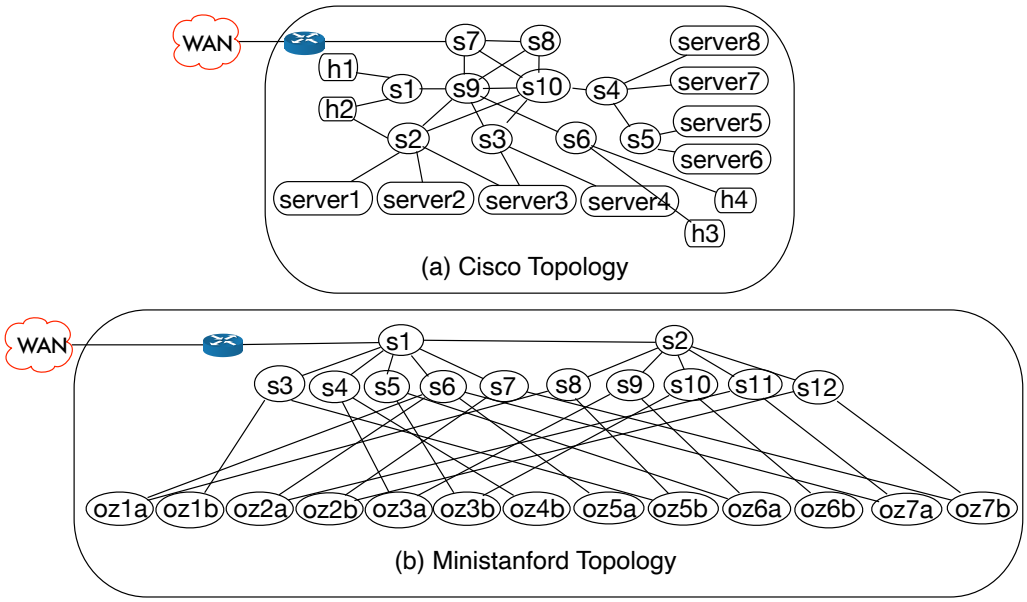


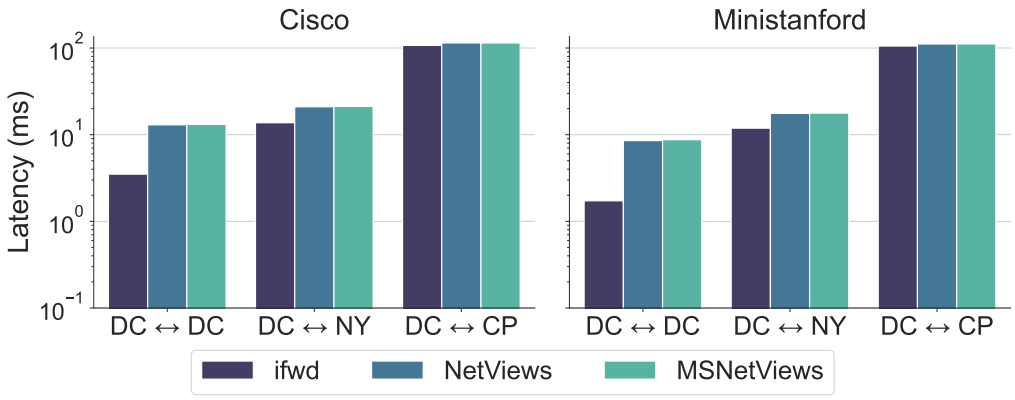
Figure 4.7: Evaluation topologies for MSNetViews.

Emulated Network Traffic: To measure overhead in cross-site communication, we program hosts to connect to other hosts only at other sites (no intra-site communication). For the Cisco topology, we initiate connections between all multi-site pairs of hosts. For the MiniStanford topology, we randomly select hosts to connect across sites to create a total of 1000 simultaneous flows. 1000 flows was the empirically determined safe limit of our experiment hardware before CPU contention of the host VM affected latencies.

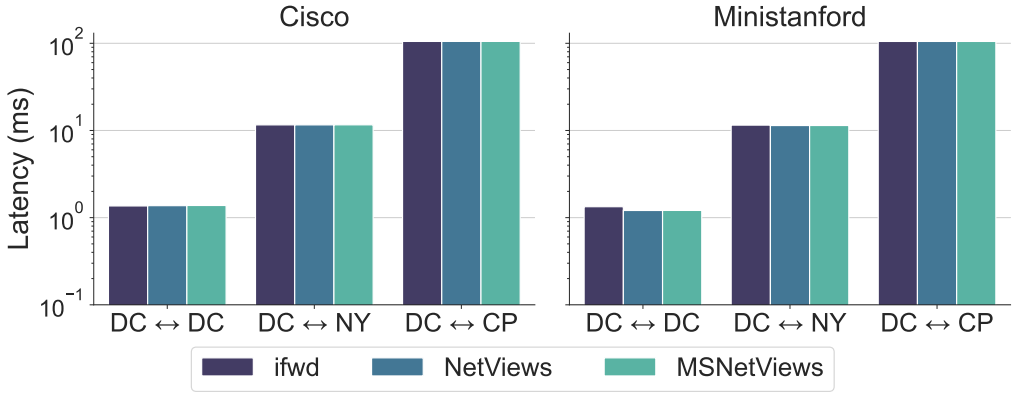
System Configuration: Similar to prior work [59, 25], we use a Docker container for each site. Within each container, an ONOS SDN controller manages an SDN network emulated by Mininet [140]. Our experiments spawned these containers on a VM on a server-class host with a AMD EPYC 7302P processor. The VM ran Ubuntu 20.04 LTS (Linux Kernel 5.4.0) and had 15 cores and 235 GB RAM. In all experiments, we measured latency using ping and throughput using perf3 (version 3.7). Each experiment ran for 60 seconds and was repeated 20 times. We present the average over these runs. Before each run, we clear the ONOS controller of any flow rules to establish a repeatable, "clean-slate" measurement. We start with a single flow, introducing a new flow every 100 ms. We summarize the experimental parameters we used in Table 4.4.

4.8.2 MSNetViews Overhead

We compare MSNetViews with standard intent forwarding (`ifwd`) and NetViews [9]. `ifwd` serves as a baseline of minimum network performance without the effects of MSNetViews. Our comparison with NetViews demonstrates multisite scalability. If MSNetViews lags in performance



(a) Average Initial Packet Latency



(b) Average n^{th} Packet Latency

Figure 4.8: Average end-to-end packet latency for MSNetViews, NetViews [9], and ifwd under three WAN latencies between sites: (1) same city (*Washington DC↔Washington DC*), same region (*Washington DC↔NY*), and global (*Washington DC↔Copenhagen(CP)*).

Table 4.4: Parameters for Performance Overhead

Parameter	Value
Total flows in MiniStanford Topology	1k
Total flows in Cisco Topology	32
Traffic pattern for experiments with 2 sites	site 1 → site 2
Wait between consecutive connections	100 ms
Same city latency (DC↔DC)	1 ms
Same region latency (DC↔NY)	11.2 ms
Global latency (DC↔CP)	105 ms

compared to NetViews, then the multi-site components would be to blame. For these experiments, there are no roaming or policy update events. Hence, the overhead comes from installing the SDN rules. As MSNetViews uses reactive SDN, forwarding rules are installed only when triggered by first packet of a new flow resulting higher overhead of the first packet than subsequent packets [9]. Our goal is to show that MSNetViews performs similar to NetViews, indicating minimal overhead for adding multisite functionality.

Before examining the experimental results, note that the the majority of network accesses in an MSNetViews environment will not result in roaming events, policy updates, or policy slicing. Rather, the most observable overhead will come from installing the desired SDN rules. As MSNetViews uses reactive SDN, forwarding rules are installed only when triggered by first packet of a new flow. Consequently, we expect the overhead of the first packet to be higher than for subsequent packets. This is consistent with the findings in Anjum et al. [9].

For all studied WAN latencies, the 1^{st}-packet latency overhead of MSNetViews over NetViews remained well under 1.2% for Cisco and 2.5% for Ministanford (see Figure 4.8a). When compared to `ifwd` on same-city (DC↔DC) traffic, MSNetViews increased 1^{st}-packet latency by 7 ms (≈5x) on the MiniStanford topology and by 9.6 ms (≈4x) on Cisco (see Figure 4.8a). In the same region (DC↔NY) , the measured overhead compared to `ifwd` was 5.8 ms to 7.5 ms (49.1% to 54.5%), and for the global scale (DC↔CP) , the overhead compared to `ifwd` was 6 ms to 7.3 ms (5.7% to 6.8%) across the two topologies. As the sites move further apart, the overhead decreases proportionately as the WAN latency dominates. Furthermore, latency overhead is limited to the 1^{st}-packet only and does not propagate to subsequent packets (Figure 4.8b). The n^{th}-packet latencies for NetViews, `ifwd`, and MSNetViews is similar, with a small overhead ($<$ 0.025 ms). The n^{th}-packet latencies for MSNetViews are 1.2 ms to 1.38 ms for DC↔DC, 11.4 ms to 11.6 ms for DC↔NY, and 105 ms for DC↔CP. In total, these results demonstrate that latency overhead will be negligible in real-world scenarios.

Figure 4.9 shows throughput results for DC↔DC and DC↔CP. For both cases, MSNetViews and NetViews are comparable, with a difference of less than 1% in the median for both Cisco and Ministanford topologies (Figure 4.9). For `ifwd` in the DC↔DC case with the Cisco topology, the

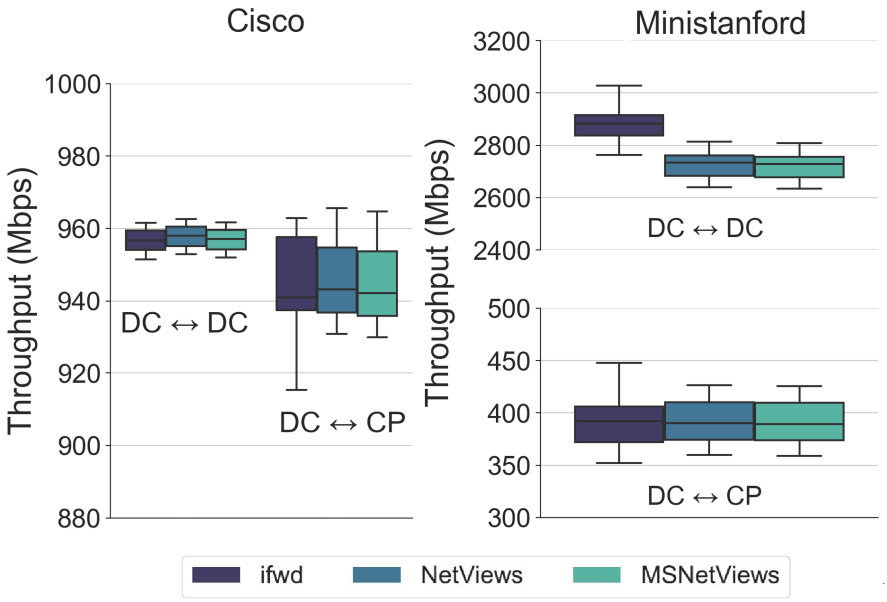


Figure 4.9: Aggregate throughput for MSNetViews, NetViews, and ifwd under two WAN latencies between sites: (1) same city (*Washington DC↔Washington DC*), and global (*Washington DC↔Copenhagen(CP)*). The scales differ for readability.

median MSNetViews throughput falls within the inter-quartile ratio of `ifwd` and the difference in medians is under 1% (Figure 4.9). For `DC↔DC` with the MiniStanford topology, MSNetViews's median aggregate throughput is approximately ≈150 Mbps below `ifwd`, a 5.3% decrease (Figure 4.9). For `DC↔CP`, the median MSNetViews throughput falls within the inter-quartile ratio of `ifwd` for both topologies. The results suggest that MSNetViews does not reduce network throughput compared to NetViews and `ifwd`.

4.8.3 Post-Roaming Stabilization

When a user roams to a new site, both local and global policy updates occur (Section 4.4). We measured the time for the policy state in all sites to stabilize to a consistent state after a user location is updated. For a worst-case estimate, we evaluated post-roaming stabilization for the global scale scenario (`DC↔CP`), which has the longest inter-site latency. Note that updates use NGAC obligations; hence do not invoke the policy checker or policy slicer.

We define *location update time* as the time difference between when the roaming user $\langle Alice, L1\rangle$ starts the authentication process at a new site, y, and when a location and identity update has been triggered in all n *relevant* sites of the enterprise (steps ❶ and ❸, in Figure 3.2). Let Auth($\langle Alice, L1\rangle, S_y$) represent the time of authentication of $\langle Alice, L1\rangle$ on site S_y, and let Trigger($\langle Alice, L1\rangle, S_x$) represent the time when the local manager of a *relevant* site x updates the location and identity of the user in the database. We define location update time

as: $\underset{x}{max}[\text{Trigger}(\langle Alice, L1\rangle, S_x)] - \text{Auth}(\langle Alice, L1\rangle, S_y)$. This measurement is the worst-case time for the user's location to be consistent at any relevant site. This time is *only* experienced when the user needs to access non-local resources (resources at other sites) immediately after authenticating. When accessing local resources, users need not wait for the update to propagate through the enterprise.

We studied the impact of two factors on the *location update time*: (1) the number of *relevant* sites in the enterprise, and (2) the number of roaming users concurrently authenticating to a new site. (3) the batching of location update events. For these experiments, we considered the MiniStanford topology.

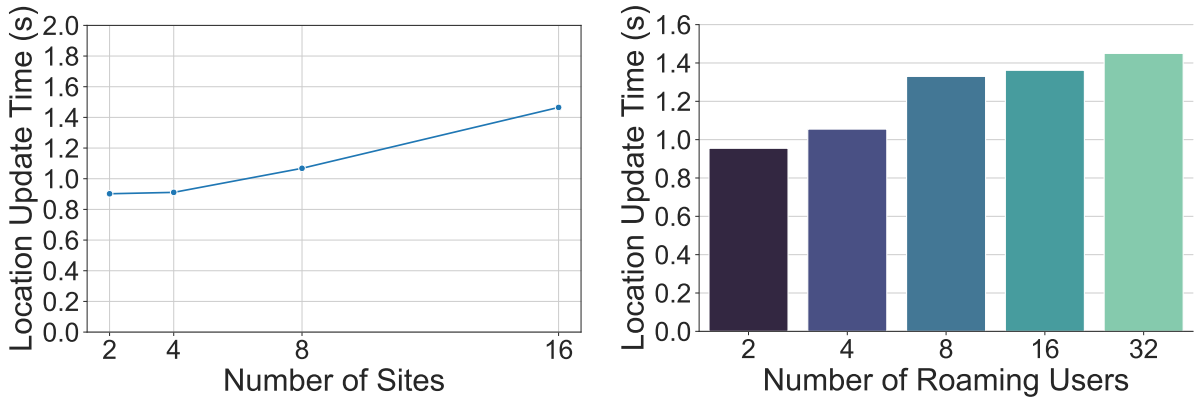
Relevant Sites: We emulated moving a single user from one site to another, and varied number of relevant sites in the enterprise exponentially from 2 through 16. As shown in Figure 4.10a, the time to complete the roaming process remained under 1.5 s across all cases. This time is *only* experienced when the user needs to access non-local resources (resources at other sites) immediately after joining a new site. When accessing local resources, users need not wait for the update to propagate through the enterprise. This time is small and has minimal impact on user experience (impact is similar to that of refreshing a page on a web browser).

Roaming Users: In this experiment, we restricted the number of sites to two, and exponentially varied the number of users moving between sites from 2 through 32. Figure 4.10b show the average location update time for each set of roaming users. The median update time for all users remained under 1.5 seconds, which is minimal impact for users joining a network after roaming from another site. This short time would cause minimal impact on user experience.

Batching Updates: The above results do not include batching optimizations that MSNetViews would use in practice. Finally, location updates can also be batched to support many more simultaneous location update events.

Methodology: To provide batching, the global manager combines all location update events received in an interval (batch interval) and sends them as one location update message to the relevant sites (see Figure 3.2). We measured the effect of 1 s and 10 s batch intervals on average location update time per roaming user. Our experiments took a two-site enterprise and exponentially increased the number of users moving between sites from 2 through 32, with a new user roaming between sites every 100 ms.

Results: Figure 4.11 presents our results. In our measurements, we include both the time taken by MSNetViews to achieve consistency across the enterprise and the wait time introduced by batching. As batching buys benefit when more updates are packed in a single batch, we observe that as the number of users increased, the average location update time per user decreased as compared to the opposite trend in experiments without batching (Figure 4.10b). For batch intervals of one and ten seconds, we observe that average location update times varied between 0.67 s to 1.4 s (Figure 4.11a) and 8.5 s to 10.4 s (Figure 4.11b), respectively. Comparing these



(a) Location update time of one roaming user as a function of number of *relevant* sites

(b) Avg. location update time per user as a function of number of users roaming between two sites

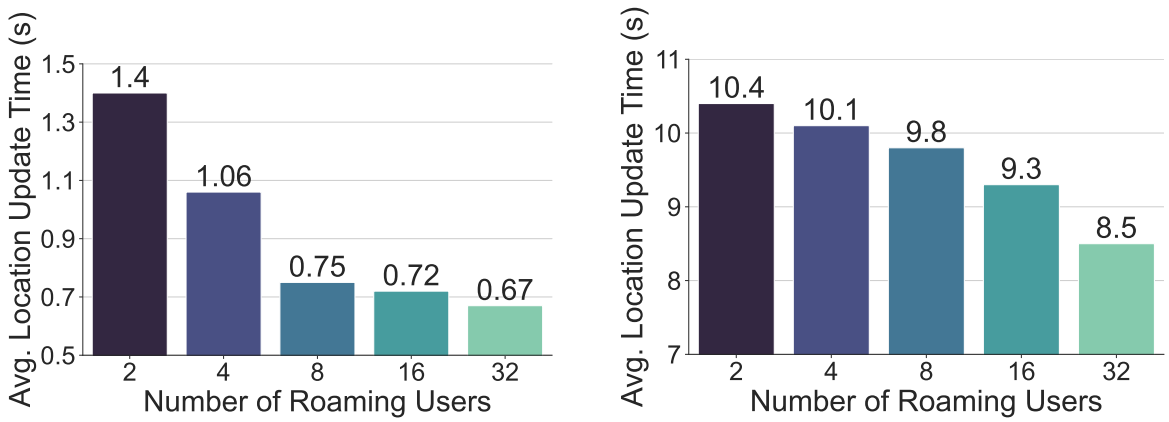
Figure 4.10: Effect of number of roaming users and number of *relevant* sites on average location update time per user for users roaming globally (between *Washington DC↔Copenhagen(CP)*). Update events are not batched.

results with non-batching results in Figure 4.10b, we note that while it took under 1.5 seconds, on average, to achieve location consistency of 32 users without batching, it took only 0.67 s for the same number of users with batch interval of one second.

4.8.4 Policy Update Performance

Administrative updates to the global policy trigger two operations that impact performance: (1) checking the policy for syntax errors—an operation performed by the policy checker (Section 4.6.2)—and (2) computing the new policy slices (Section 4.5)—an operation performed by the policy slicer. Note that, the above mentioned operations are not initiated for a user roaming events. We measured the time required to complete policy checking and slicing independently for different policy configurations.

Methodology: Table 4.5 summarizes the parameters used for experiment setup. We evaluate the performance of *global manager in the event of administrative policy update* by experimenting individually on the policy checker and policy slicer components. We used the policy graph of the MiniStanford topology for five component sites, with a host count varied within the range of 100 to 10000. We run this experiment in the same VM host as our previous experiments. To generate self-consistent policies, we rely on the *random policy creation* algorithm as in prior works [9, 102]. This algorithm creates a binary tree-like connectivity among host nodes (user-device pairs and objects) and policy elements. The height of the policy graph (binary tree) and host nodes determines the policy graph complexity (number of policy nodes). The number of policy nodes is determined by calculating $(2^{h+1} - 1) \times n$, where n is the number of hosts and



(a) Batch Interval of 1 sec

(b) Batch Interval of 10 sec

Figure 4.11: Average location update time per user with batch processing at two different batch intervals as a function of number of users roaming globally (between *Washington DC↔Copenhagen(CP)*)

Table 4.5: Experiment Setup for Policy Update Overhead

Parameter	Value
Total number of sites	5
Local site topology	MiniStanford
Total number of hosts	$100 \rightarrow 10000$
Total number of height	$1 \rightarrow 2$
Total number of policy node	$300 \rightarrow 70000$
Total number of update-variant	10×50

h is the height of the policy graph. The maximum value of the graph height was two for our experiments.

We created a *policy variant generator*, which can create graph variations from a supplied base policy graph using one or more randomly selected update operations. We considered ten types of update operations among possible fourteen types (e.g., add user node, delete association). We avoided the delete operation on attribute nodes and assignment relations for the ease of experiment, as these delete operations can produce a disconnected policy graph. Finally, we generated 50 policy variants of each type of update operation for each run.

Results: As shown in in Table 4.6, the policy graph complexity is a significant factor in policy checking and slicing delay. This is expected because the policy checking and slicing algorithms traverse the policy at-least twice. Note that this experiment uses the worst-case algorithm (binary tree) for policy generation.

Shown in Table 4.6, the average delays for both the policy checker and slicer follow similar trends and are marginal even for a significant policy graph with 10,000 host nodes and 30,000

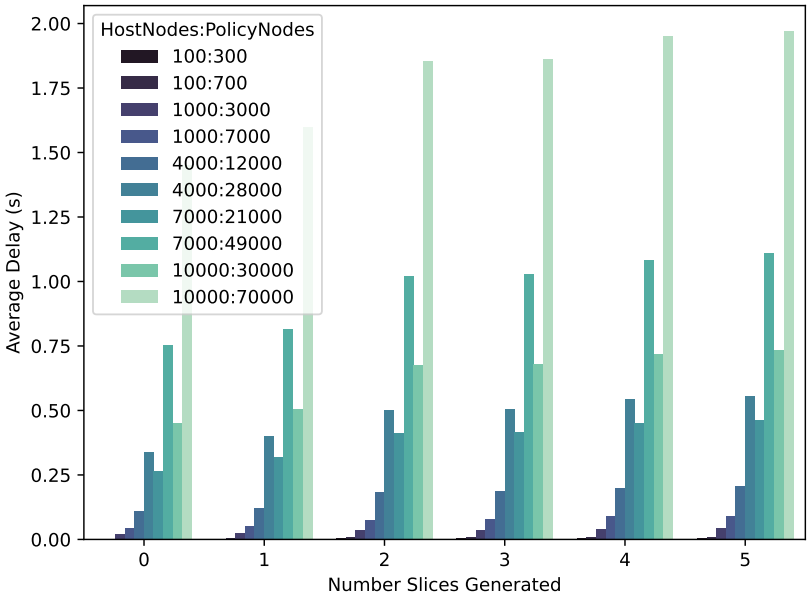


Figure 4.12: Effect of Number of Slices Needed to be Generated for Policy Updates.

Table 4.6: Effect of Policy Graph Complexity on Average Policy Checking and Slicing Delay

Host No.	Policy Node No.	Average Delay (ms)	
		Policy Checker	Policy Slicer
100	300	3	6
100	700	6	9
1000	3000	25	38
1000	7000	62	81
4000	12000	151	189
4000	28000	452	516
7000	21000	388	428
7000	49000	1153	1024
10000	30000	654	688
10000	70000	2441	1883

policy nodes (0.64 seconds and 0.68 seconds, for checking and slicing, respectively). We also analyzed if the operation types affect the policy update overhead. The standard deviation for the 10 operation types varies from 6 to 10 ms, and for policy slicing it varies from 25 to 35 ms for different numbers of host and policy nodes.

For each update, MSNetViews regenerates policy slices; however, it is not necessary to generate new slices for sites unimpacted by the change (Section 4.5.3). Figure 4.12 demonstrates that the policy slicing itself does not have much impact on the performance. On average, policy slicing occupies 30% to 40% of the total delay; the rest of the time is needed to load the new policy graph and find the differences from the base graph. This delay increases significantly with the increasing complexity of the policy graph. We demonstrate this in Figure 4.12.

4.9 Discussion

Table 4.7: NIST Network Requirements to Support ZTA

No.	Requirement	MSNetViews
1.	Enterprise assets have basic network connectivity	Yes
2.	Enterprise can observe all network traffic	Yes
3a.	The enterprise must be able to distinguish between what assets are owned or managed by the enterprise	Yes
3b.	The enterprise must be able to distinguish between the devices' security postures	No
4.	Enterprise resources should not be reachable without accessing a PEP	Yes
5.	The data plane and control plane are logically separate	Yes
6.	Enterprise assets can reach the PEP component	Yes
7.	The PEP is the only component that accesses the policy administrator as part of a business flow	Yes
8.	Remote enterprise assets should be able to access enterprise resources without needing to traverse enterprise network infrastructure first	out-of-scope
9.	The infrastructure used to support the ZTA access decision process should be made scalable to account for changes in process load	Yes
10.	Enterprise assets may not be able to reach certain PEPs due to policy or observable factors	Yes

MSNetViews is not designed as a full zero trust solution for on-premises devices. Rather, our goal is to provide novel foundations that solve key challenges. We now discuss the engineering enhancements needed to extend MSNetViews to provide a full Zero Trust Architecture (ZTA) solution.

There are many industry and government interpretations of ZTA. Due to the lack of a formal ZTA definition, we cannot provide a formal proof for MSNetViews. NIST defines ten "Network Requirements to Support ZTA" in Section 3.4.1 of their special article on Zero Trust Architecture [123]. Table 4.7 shows how MSNetViews handles those requirements. MSNetViews satisfies all but two of the ten requirements. Basic connectivity features also allow enterprise-owned or managed resources to reach the Policy Enforcement Point (PEP) with the necessary scope to enforce restrictions. Note that MSNetViews's scalability depends on SDN technologies, which have been shown to scale to large networks [153, 44, 128]. One of these requirements out-of-scope: NIST states enterprise resources should be accessible without needing to traverse enterprise network infrastructure. While this applies to business applications, the on-premises

devices protected by MSNetViews cannot be moved off-premises. MSNetViews partially addresses the second requirement it fails to satisfy, which states "the enterprise must be able to distinguish between what assets are owned or managed by the enterprise and the devices' current security posture." MSNetViews does capture which devices are managed by the enterprise; however, it does not incorporate the current security posture of devices. Existing zero trust solutions accomplish this requirement using device attestation and behavioral analytics. Such information can be incorporated into MSNetViews's policy decision and is a straightforward engineering effort for future work.

4.10 Summary

While reactive SDN technologies provide a promising foundation for realizing zero trust goals within on-premises enterprise networks, they are limited to single networks, requiring enterprises with multiple sites to manually ensure consistency of security policy across all sites. This chapter presented MSNetViews, an system that extends a single, globally-defined and managed, enterprise network security policy proposed in NetViews to many geographically distributed sites. Each site operates independently and enforces a site-specific policy slice that is dynamically parameterized with user location as employees roam between sites. Our prototype implementation shows that policy state inconsistencies that result from user roaming settle quickly, well below two seconds on average after a user roams to a new site. In presenting MSNetViews, we demonstrate that SDN can not only provide an invaluable primitive for achieving zero trust within a single enterprise location, but also across many geographically distributed locations.

Chapter 5

HoneyRoles

In Chapter 3 and Chapter 4 we demonstrated how reactive forwarding can enable granular least privilege policy enforcement to defend against the threat of compromised host. However, in many recent events, we have seen that the forwarding network devices themselves can be compromised and be prime targets [66, 111]. These computationally capable compromised forwarding devices can allow an attacker to do targeted passive and active reconnaissance. This type of backdoor also provides ample capability for the attacker to propagate and compromise other hosts and critical infrastructures. The criticality here is the presence of these backdoors can be very subtle and difficult to detect (e.g., Snowden [134], political espionage [92]). This chapter addresses the challenge of compromised forwarding devices while protecting enterprise employees functioning in high-value roles.

5.1 Introduction

Enterprises heavily rely on the security of their networks. These networks often consist of a wide variety of computing resources, including desktops, laptops, servers, routers, and switches. The resources support a range of activities by different types of users performing actions as different roles (e.g., IT administrators, C-suite executives, and finance personnel) [26]. By compromising one or more of these resources, an adversary may cause significant harm to the enterprise. For example, it may steal credentials or access systems with the goal of exfiltrating sensitive information such as intellectual property and customer information, or modifying data such as source code repositories and payment systems.

The first phase of network infiltration is reconnaissance. Traditional reconnaissance techniques such as port scanning are *active*, and the current state-of-the-art network defenses have become highly tuned to identify them. However, *passive* reconnaissance by compromising packet forwarding devices and inspecting network flows to identify the existence and behaviors of client and server hosts is becoming increasingly feasible. Specifically, as packet forwarding devices such

as routers and switches become more complex, they become more prone to compromise [152, 32]. These targets include emerging Software Defined Networking (SDN) switches, which provide much broader and more flexible functionality [134, 141, 18, 11]. Prior solutions [120, 106] seeking to defend against malicious forwarding devices are not directly applicable for SDN devices [160]. Furthermore, SDN data plane defenses mostly concentrate on forwarding verification and other active attacks (e.g., packet delaying, tampering, dropping) [39, 81, 131, 31, 93].

The goal of this chapter is to protect enterprise employees acting in high-value roles such as IT administrators, C-suite executives, and finance personnel. We are particularly interested to (1) deceive adversaries by perturbing the network traffic information gained through passive reconnaissance, and (2) dissuade an adversary from acting on observed information (e.g., performing active reconnaissance or an attack). Our vision is to build metaphorical "haystacks" around the network activities of these individuals. The introduced network traffic perturbs reconnaissance, and if the adversary acts on the wrong intelligence, it will be detected with high probability, which will in effect dissuade the adversary from acting.

In this work, we propose HoneyRoles, which uses *honey connections* to deceive adversaries using compromised packet forwarding devices for *passive reconnaissance*. HoneyRoles coordinates honey connections by modeling fake hosts that are organized into roles corresponding to organizational functions of client hosts. HoneyRoles performs integrity validation of honey connections such that they act as "canaries" for attacks against network clients. In the event that an adversary modifies or blocks a honey connection, HoneyRoles detects the adversary's existence and statistically identifies any compromised forwarding devices.

We evaluate the security of HoneyRoles' defender-attacker environment using a probabilistic model checker (PRISM [89]). This simulation assumes an alert has been raised and measures the accuracy of detecting the location of compromised switches. For a simulated Fat-Tree network topology with 50 real and 50 fake hosts and 1 compromised switch, we show that HoneyRoles consistently ranks the compromised switch as most suspicious. In the same environment with two compromised switches, we show that HoneyRoles consistently ranks at least one of the compromised switches as most suspicious. The second compromised switch is also usually highly ranked, depending on its function.

We additionally used Mininet to emulate the Fat-Tree topology with 50 real and 50 fake hosts. When measuring the pairwise request completion time between real hosts and servers, we observed that HoneyRoles has a small impact on network request completion time for a moderately loaded network (1 request per second per host). With a thorough experiment, we have seen that 90% of hosts observe less than 14% overhead in request completion time.

This chapter makes the following contributions:

- *We introduce role-based deception as an enterprise network defense.* HoneyRoles conceals the identity of critical client hosts and creates uncertainty for an adversary residing in one or more compromised packet forwarding devices.

- *We use honey connections to deflect and detect en-route manipulation of client network traffic.* HoneyRoles uses statistical inference to identify any compromised network device.
- *We evaluate the security of HoneyRoles's defender-attacker environment using a probabilistic model checker.* HoneyRoles consistently tracks network events and successfully ranks the switches in terms of suspiciousness.

5.2 Problem

Targeted attacks [41, 92] and threats to enterprise network infrastructure [66, 142] continue to increase. Such attacks often begin with a foothold for reconnaissance. Historically, footholds have been client workstations. However, network packet forwarding devices such as routers and SDN switches are becoming prime targets as they offer a valuable vantage point for reconnaissance and their increased complexity leaves them more prone to compromise.

Once a foothold is established, the adversary performs reconnaissance to identify targets that most profitably support its goals (e.g., to take over the account of an IT administrator or C-suite executive). From the vantage point of a compromised network switch, the adversary can perform various en route network traffic attacks that strategically and selectively target high-value clients at critical times. For example, it could inject malicious JavaScript into Web pages as they are returned from Web servers, or it could use SSL-stripping to eavesdrop on traffic and steal credentials. Existing defenses such as HSTS have seen limited deployment [87], in part because many developers do not understand how to use HSTS correctly, resulting in critical information such as login cookies being leaked. For networks that include mobile devices, Luo et al. [98] found that popular mobile web browsers failed to fully support HSTS and were left open to clickjacking attacks. Additionally, Krombholz et al. [88] showed that TLS deployment is far too complex, leading to large numbers of incorrect HTTPS deployments. Other attacks include redirecting client traffic to malicious servers or simply blackholing the traffic to keep a target from performing a critical task (e.g., monitoring IDS logs). If done strategically and sparingly, such manipulation can fall under the detection thresholds of existing defenses [39, 31, 131].

Such attack activity can be broken down into three phases. *(1) Passive reconnaissance:* the adversary passively intercepts and tracks the communications of different organizational entities to identify the target roles' probable locations. Other than forwarding collected data for further analysis, the adversary does not leave a trace for the defender to identify suspicious activity. *(2) Active reconnaissance:* the adversary may perform a different type of active interception for pinpointing the target and increasing the confidence it has about the information. Such activities may be detected by the defender; however, the adversary still does not disrupt communication. *(3) Active attack:* the adversary has gained adequate confidence for target systems and decides to attack a client's network traffic. Even if such activities raise an alarm, the adversary's location within the network may still be difficult to locate.

The three-phase attack plan described above demonstrates the danger of reconnaissance as an important precursor to sophisticated attacks. With information about the users, devices, and services on a network it is possible to design an attack strategy that minimizes the risk of detection. For example, armed with information gathered passively, an adversary may realize its current foothold is unable to contact a sensitive server without triggering an alarm, resulting in it pivoting its foothold in the network to a device or user that can access the server. For this reason, it is crucial to defend against network reconnaissance.

Threat Model & Assumptions: The goal of the adversary is to identify high value targets, learn enterprise secrets (e.g., intellectual property, customer data and credentials), and modify data en route to high-integrity servers (e.g., software code repositories, payment systems). To do so, an adversary may target administrative systems, or connections to them, to gain access to target systems. We assume the adversary is able to compromise one or more packet forwarding devices in the network. From the vantage point of a forwarding device, the adversary can view, analyze, and modify all packets that flow through it. We do assume that not *all* of the forwarding devices are compromised, and that the defender can incrementally replace or refresh devices as they are detected.

We assume the adversary has some, but not all knowledge of the hosts in the network. For example, we assume the IP addresses of important servers (e.g., Admin and Finance Servers) are known, based on other available information (e.g., DNS information). However, we assume the adversary does not know the IP address and other details of workstations that perform specific organization roles (e.g., the IT Admin workstation). Additionally, by compromising an SDN switch, the adversary has access to the SDN southbound network and hence can attempt to forge forwarding rules (e.g., OpenFlow messages) in the corresponding switch. Finally, in order to achieve its goals, the adversary seeks to remain *undetected*.

Our trusted computing base (TCB) includes the system defender (SDN controller or a separate trusted server) and the southbound network between the SDN controller and SDN switches. As such, we assume the SDN switches are configured to either use out-of-band communication, or in-band communication protected by TLS. We do not blindly assume that SDN switches are trustworthy. Similarly, HoneyRoles trusts its host agents running on workstations and servers. Finally, we assume the topology has a sufficient number of redundant forwarding paths for the ease of dynamic path management, discussed in Section 5.4.

5.3 Overview

HoneyRoles seeks to use deception to mitigate the threat of compromised packet forwarding devices (e.g., switches and routers). From the vantage point of a packet forwarding device, an adversary can perform passive reconnaissance to identify high-value client hosts (e.g., IT administrators, C-suite executives, and finance personnel), active reconnaissance (e.g., selective

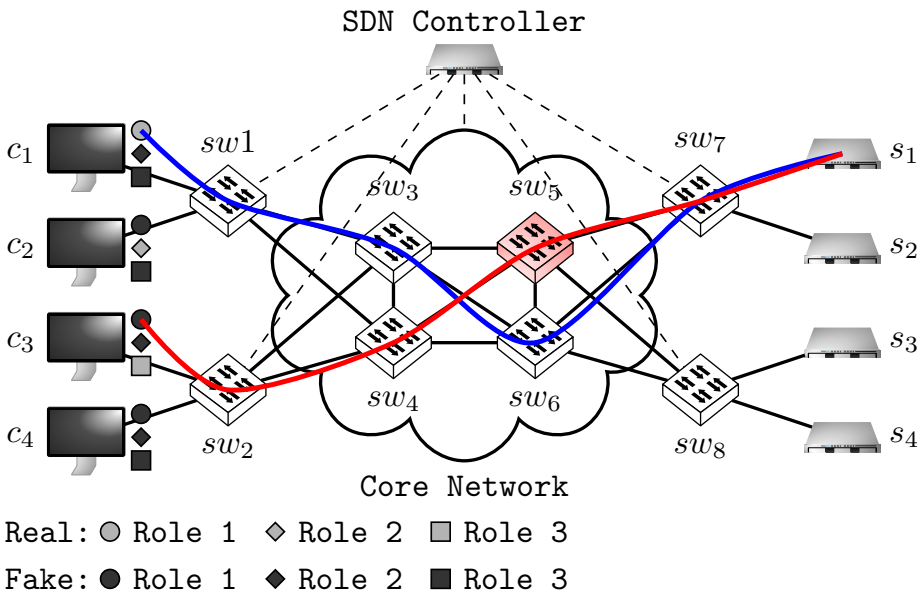


Figure 5.1: Overview of HoneyRoles

probing or rerouting), and perform en route traffic attacks (e.g., injecting content, SSL-stripping, blackholing). Our vision is to introduce honey network traffic that (1) deceives the adversary by building metaphorical "haystacks" around the network activities of high-value client hosts, and (2) dissuades the adversary from acting on information in real network traffic for fear of being detected. Achieving this vision requires overcoming the following research challenges:

C1 (Detection): *Compromised packet forwarding devices are difficult to detect.* An adversary performing passive monitoring will not produce detectable actions until it attempts an attack (possibly months after compromise), at which point it may be too late to detect a compromise.

C2 (Exposure): *The adversary may have knowledge of some enterprise network components.* Information from DNS and publicly accessible websites [104, 139] make hiding the identity of servers futile. Servers receive a disproportionate amount of inbound connections, allowing an in-network adversary to distinguish between clients and servers, which may limit the effectiveness of moving-target defenses [121, 34].

C3 (Visibility): *The adversary may be aware of the deception system.* Naïvely sending honey traffic is not effective if the adversary is aware of the defense. External events (e.g., stock market changes and DDoS attacks) can cause certain high-value client hosts to act predictably.

Figure 5.1 overviews the high-level intuition behind HoneyRoles. The figure depicts four client hosts (c_1-c_4) and four servers (s_1-s_4) connected by a network topology with redundant

links and switches. A network administrator partitions each client host based on *organizational roles*. In the figure, c_1 is in `Role 1`, c_2 is in `Role 2`, c_3 is in `Role 3`. Host c_4 is not assigned to any role. HoneyRoles then installs software agents (honey agents) on client hosts that produce fake network traffic. Each client host is assigned at least one honey agent, each of which is assigned one of the organizational roles. Some physical hosts may run multiple honey agents.

HoneyRoles uses hosts with honey agents to establish *honey connections* with the real servers. Using *honey connections*, honey agents establish *new* application-layer protocol sessions with the servers (simply replaying network traces would be detectable). HoneyRoles uses Software-Defined Networking (SDN) to dynamically select random forwarding paths between client hosts (both honey and real) and servers. Figure 5.1 shows two forwarding paths. The blue path (c_1-s_1) containing real traffic avoids the compromised switch (sw_5), and the red path (c_3-s_1) containing fake traffic passes through the compromised switch (sw_5). Note that the blue path could have just as easily passed through the compromised switch. The goal of deception is to provide the adversary with fake information such that it does not know what information to believe.

HoneyRoles addresses the *Detection* challenge by increasing the frequency at which a given compromised packet forwarding device will see network traffic that may be viewed by the adversary as "valuable". If the adversary guesses wrong enough times and performs an en route traffic attack on a honey connection, HoneyRoles statistically identifies the compromised forwarding device. For example, a switch near client host c_4 (Figure 5.1) may not normally see network traffic to a domain controller. However, honey connections from a honey agent on c_4 provide the delusion that domain controller traffic can go through that switch. In fact, this design choice enables HoneyRoles to not merely overcome but embrace the *Exposure* challenge. By not obfuscating the network identities of high-value servers, HoneyRoles uses honey connections as bait.

HoneyRoles addresses the *Visibility* challenge through its use of organizational roles to parameterize the creation of honey connections. For each role, an administrator specifies a *role profile*. The profile defines: (1) the number of real hosts, (2) the identities of the real hosts, (3) the number of honey agents, (4) the locations and identities for the honey host agents, and (5) the set of target servers S that are relevant to the role (e.g., domain controller, HR server). HoneyRoles assumes the adversary would attempt to identify target client hosts based on their connections to the high-value servers. Therefore, HoneyRoles monitors the network activity between real clients and corresponding high-value servers. It uses these traffic patterns to automatically configure the honey host agents to send honey connections to the target servers with similar request rates. Assume there are r real client and h honey host agents in same target role. Hence, the adversary has a $\frac{r}{r+h}$ chance of correctly identifying a real client host. Importantly, HoneyRoles does not provide any signal on detection (no change of strategy) toward the adversary.

Finally, to ensure the adversary cannot distinguish a honey host from a real host, HoneyRoles assumes an ambient network traffic generator to represent the general activity that a given user may perform [146, 135, 22]. We note that these prior works are simply examples. Designing and evaluating network traffic generators that can evade detection from modern machine learning algorithms is an orthogonal challenge. HoneyRoles would directly benefit from any advancements in this area.

5.4 Design

This section discusses three considerations in HoneyRoles: (1) *honey connections*, including how they are managed by the software agents and how they are coordinated by the HoneyRoles controller; (2) dynamic *forwarding path management* to distribute honey connections across many potentially compromised switches and help statistically identify compromised switches; and (3) the *belief maintenance system* within the HoneyRoles controller, which is used to rank switches based on their probability of being compromised.

5.4.1 Honey Connections

Honey connections provide the primary form of deception in HoneyRoles. For expository reasons, we describe how to create honey connections for a single role; it is straightforward to extend the design to an arbitrary number of roles.

Role Profile

For each role, the administrator defines a fixed set ID_r of real hosts matching their organizational roles. The administrator defines for each role a honey host factor, $\alpha \geq 0$, which yields the size of the set ID_h of honey hosts for the specific role. Specifically, $|ID_h| = \lceil \alpha \cdot |ID_r| \rceil$. We expect a typical deployment will include at least as many honey hosts as real hosts ($\alpha \geq 1$).

HoneyRoles identifies each host via a 5-tuple: $<$ `ip`, `mac`, `type`, `role`, `switch` $>$, where `ip` is the host's IP address, `mac` is the host's MAC address, `type` indicates if the host is real or honey, `role` specifies the host's organizational role (e.g., IT administrator), and `switch` specifies the network switch to which the host is attached. Both real and honey get unique identities. The `ip`, `mac`, and `switch` are fixed for real hosts and are randomly assigned by HoneyRoles for honey hosts. Of these values, the `switch` has the most impact on the utility of honey connections, as it determines where in the network the honey host exists, and hence the other switches that honey connections to or from this switch will likely traverse. The honey agent (host) composition and assignment is further discussed in Section 5.4.1.

Finally, the role profile contains a set of target servers S, each associated with the organizational role. Conceptually, S defines the set of servers that users in a role connect with to

perform their duties. For example, for an IT administrator role, S may include a domain controller, a centralized VM management server, and a configuration management system server. For each server $s \in S$, the network administrator specifies information for valid connections (e.g., an unprivileged user and associated credentials) so that the size and frequency of packets in TLS-protected connections are indistinguishable from the connections generated by real hosts.

Honey Agent

We envision that honey agents will reside on the same physical hardware as real hosts to reduce capital expenditures required for deploying HoneyRoles. However, network administrators can deploy hosts without users, e.g., decommissioned computers, that run only honey agents.

The honey agent needs to be a privileged process capable of using distinct IP and MAC addresses. This is achievable using either operating system virtualization or containerized environments. For example, Qubes OS uses a hypervisor to containerize multiple distinct execution environments. It provides a flexible and modular networking environment that can bridge virtual interfaces to different environments. Alternatively, for non-hypervisor hosts, the honey agent could be deployed as a container. Since our performance evaluation in Section 5.6 uses Mininet [140], we emulate the existence of a honey agent by creating extra Mininet hosts (with individual IP addresses) tagged as honey hosts.

Honey Agent Coordination

The HoneyRoles controller coordinates the honey connections sent by honey agents using TLS-protected heartbeat messages. It is important that heartbeats are sent on regular intervals and are statistically similar in size, as they are sent through the data plane and are observable by adversaries. Heartbeat messages are sent to real hosts to prevent the adversary from using heartbeats to identify honey hosts.

The purpose of the heartbeat messages is to parameterize the creation of honey connections. To that end, each heartbeat contains the following information: (1) destination information (MAC address, IP address, transport-layer port), (2) number of RREs (Request Response Exchange), (3) RRE interval, (4) application-layer protocol information, and (5) estimated timeout. The generation of believable honey connections additionally requires realistic application-layer content or information. The application-layer protocol information depends on the type of protocol (e.g., SMTP, FTP, HTTP). For example, HTTP/HTTPS connections may require a URL, cookies, and username/password pairs. Other application-level information can be Gmail cookies, protocol payloads (i.e., email bodies), passwords for unencrypted protocols (e.g., SMTP, POP, IMAP). For simplicity, our implementation considers only HTTP and HTTPS traffic.

Capturing Real Traffic Profiles: As described in Section 5.3, a key idea of HoneyRoles is that honey connections for a given role follow the traffic patterns of that role. Existing traffic tracing and monitoring tools [51, 114, 58] use multiple network sensors distributed throughout a network. We achieve a similar capability using OpenFlow's flow-level statistics collection mechanism [144, 137]. Our implementation leverages this information within the ONOS SDN controller. We leverage the OpenFlow control messages (e.g., PacketIn, FlowMod, FlowRemoved, FlowStatistics) to capture the near-realtime traces of real host connections.

Replicating Real Traffic Profiles: Our implementation does not include ambient network traffic, but focuses on dynamically turning captured real traffic profiles into honey connections. The role profile (Section 5.4.1) defines a set of target servers S that are relevant to the tasks of a given role. HoneyRoles generates honey connections of a specific role by observing the network connections between the real hosts ID_r of that role and the corresponding servers in S. As in Harpoon [135], HoneyRoles parameterizes traffic generation based on the following information for each time interval: (1) the *source and destination addresses*; (2) the *payload size* for each source-destination pair; (3) the average *number of active sessions* between each source-destination pair; (4) the *time duration* based on an empirical distribution of time between consecutive connections as well as the inter-arrival time; and (5) *header information* based on the common values such as MAC address, protocol, and port.

Honey Agent Reports

A honey agent sends reports as heartbeat responses. Note that real hosts must also send reports (without meaningful content) to make them indistinguishable from honey hosts. At a high level, a honey agent report provides a status update on the honey connections specified in previous heartbeats. Each alert included in a report specifies: (1) total number of requests sent, and (2) alert details (e.g., average delay, number of dropped request, attack type).

Our implementation detects two attack types: SSL-stripping and blackholing. SSL-stripping occurs when the victim first visits the HTTP version of a website. Normally, the server will redirect the web browser to the HTTPS version of the website. However, an en route network adversary can suppress the redirection to keep the victim using HTTP URLs, potentially revealing passwords or other security-sensitive information. To detect SSL-stripping, HoneyRoles uses honey connections that simulate the user entering just the domain name into the URL bar of the web browser. If the honey agent does not receive the expected redirect to the HTTPS version of the web page, an alert is reported.

Network blackholing occurs when an in-network adversary prevents packets from reaching their destination. For example, an adversary may wish to prevent an IT administrator from accessing a network logging server while it is performing an attack. To detect blackholing, HoneyRoles simply sends honey connections to the important target servers. If a connection exceeds a pre-specified timeout period, an alert is reported. However, normal network congestion

and load at the target server can also cause honey connections to time-out. Therefore, the belief maintenance system (Section 5.4.3) must take care when using alerts of this type.

5.4.2 Forwarding Path Management

HoneyRoles dynamically changes the forwarding path from clients to servers to distribute honey connections across potentially compromised switches. The dynamic forwarding path helps to identify the location of a compromised switch. Since the goal of the adversary is to distinguish between real and honey connections, it is important to minimize the differences between them. Therefore, HoneyRoles does not differentiate real connections from honey connections when changing forwarding paths.

A dynamic forwarding path selection distributes packets in honey connections across more switches. To understand how the dynamically forwarding path helps identify the location of a compromised switch, consider a collection of alarms raised for honey connections between client c_1 and server s_1 (Figure 5.1). If the honey connections always traverse the same set of network switches, it is difficult to determine which switch is compromised. However, if the forwarding path differs for each alarm, the intersection of the forwarding paths can be used to isolate a compromised switch. The belief maintenance system in Section 5.4.3 uses this intuition.

HoneyRoles builds upon the OpenFlow SDN protocol to perform dynamic forwarding paths. A key component of all SDN controllers (e.g., ONOS) is a reactive forwarding path algorithm that determines the best path from a source to a destination. Network topologies commonly have redundant links and switches (e.g., Figure 5.1). We observe that given a network topology with sufficient redundancy, there will be multiple optimal (or slightly non-optimal) paths within each pair of source and destination. We change the path selection logics of the forwarding path algorithm, which also avoids forwarding loops and potentially react to network congestion.

HoneyRoles defines network flows as a 5-tuple: source IP address (s_{ip}), source transport-layer port (s_{port}), destination IP address (d_{ip}), destination transport-layer port (d_{port}), and transport-layer *protocol* (i.e., TCP or UDP). Whenever a new connection (honey or real) is set up by a source-destination pair, a PacketIn message (request for setting up a forwarding path) is sent to the controller by the edge switch connected with the source host. HoneyRoles's reactive forwarding application determines a maximal set of disjoint paths. Depending on the system requirement, this application can consider optimal disjoint paths only, or both optimal and non-optimal disjoint paths, or tolerate a certain percentage of overlap. From the set of possible forwarding paths, HoneyRoles selects a path using uniform random distribution. Even if the defender suspects compromised switches on a certain path, it should not set a priority in the selection process, as this may be detected by the adversary, thereby revealing some of the defender's knowledge.

Given a topology with p disjoint paths (both optimal and non-optimal), the probability of selecting a certain path is $1/p$. At a given time t, there are r real and h honey connections for a

given target server. If there is a compromised switch in only one disjoint path, the probability that the adversary will be able to scan a real connection is $\frac{r}{p(r+h)}$. Consequently, combining the dynamic forwarding and honey connections, HoneyRoles builds a dense haystack around the real connections, making passive reconnaissance harder.

5.4.3 Belief Maintenance System

The goal of the belief maintenance system (BMS) is to alert the system administrator about the existence of an adversary, as well as potential locations of compromised switches. However, it does not seek to precisely determine a specific switch or set of switches that are compromised. Instead, the BMS ranks switches based on a level of suspiciousness. The goal is to ensure all compromised switches are among the most suspicious ones in the ranked list. The BMS can reside on the SDN controller or on a separate server.

As discussed in Section 5.4.1, detection of adversarial activity and alert generation is performed by the honey agents. Recall that HoneyRoles uses both role-based honey connections and dynamic forwarding paths to entice the adversary into acting on false information. HoneyRoles cannot be certain about the network's adversarial state. For example, some alarms (e.g., packet dropping) can be generated from either network failure or adversarial activity. Furthermore, even for true positives for a given forwarding path with n switches, there is only a $\frac{1}{n}$ chance that a given switch is the source of the alarm. Therefore, the BMS maintains an updated mapping between the honey connections and the corresponding forwarding paths and uses alarms from honey agent reports to update its belief of suspiciousness for each switch.

The BMS updates its current belief for each switch after each discrete time interval γ. That is, if the current time is t, the next update will occur at $t + \gamma$. The BMS uses the γ period to collect statistics for the interval, after which the reports can be discarded. For each switch s_k, the BMS calculates a_k and c_k for the time interval. Here, a_k is the number of alarms received for forwarding paths that include switch s_k and c_k is number of honey connections forwarded by s_k. The BMS then calculates a risk factor $r_{k,t} = \frac{a_{k,t}}{c_{k,t}}$ for switch k on a specific time t. It computes an overall risk factor $R_{k,t}$ for switch s_k using exponential moving average (where $R_{k,0} = r_{k,0}$):

$$R_{k,t} = \beta \cdot r_{k,t} + (1 - \beta) \cdot R_{k,t-\gamma} \tag{5.1}$$

For convergence, $0 < \beta < 1$. To reduce the weight assigned to the current time interval, for our experiments in Section 5.5, we use $\beta = 0.2$; however, we have experimented with other values of β (≤ 0.5) and anecdotally found similar results. Algorithm 2 summarizes the process of belief maintenance.

The BMS creates a ranked list of switches based on their level of suspiciousness (higher $R_{k,t}$ means higher likelihood of being compromised). This list is a useful resource for the network administrator for remediation or reconfiguration.

Algorithm 2 Belief Maintenance System

```
1: procedure BeliefMaintenance(t)
2:     #Risk update using Honey Notification
3:     Initialize a_{k,t} & c_{k,t} to 0, for all switch s_k
4:     for each entry e ∈ Honey Notification at time t do
5:         P_{s,d} ← getForwardingPath(e)
6:         for all k ∈ P_{s,d} do
7:             Increment c_{k,t}
8:             if any ATTACK logged in e then
9:                 Increment a_{k,t}
10:    for all connected switch s_k do
11:        Update r_{k,t} & R_{k,t}
```

5.5 Security Analysis

HoneyRoles creates deception using honey connections from honey hosts representing different enterprise roles. In this section, we use the PRISM probabilistic model checker to characterize HoneyRoles's effectiveness against an adversary that is aware of the existence of a HoneyRoles deployment. Note that this evaluation assumes HoneyRoles has raised an alarm. The evaluation is designed to determine how well HoneyRoles can identify the compromised switch. Recall that our goal is for compromised switches to be ranked as one of the most suspicious. We begin by presenting our implementation of HoneyRoles within PRISM and then present the results of the simulation.

5.5.1 PRISM Model

Probabilistic model checking uses a model construction that represents the behavior of a system over time, i.e., the possible states that the model can be in, the transitions that can occur between states, and information about the likelihood of these transitions [3]. It can provide an approximate value of a certain parameter by calculating all possible system paths. The PRISM [89] probabilistic model checker supports three model types: Markov decision processes (MDPs), discrete-time Markov chains (DTMCs), and continuous-time Markov chains (CTMCs). We chose to use DTMC as it is more realistic for our model to consider time as discrete steps for maintaining the belief state of each switch.

A PRISM model is constructed as the parallel composition of its modules. The behavior of each module is described by a collection of guarded commands.

$$[\,]\ \mathit{guard} \rightarrow p_1 : u_1 + \ldots + p_n : u_n;$$

Here, the guard *guard* is a predicate over model variables. Each update action u_i describes

Table 5.1: HoneyRoles Configuration in PRISM

	Environment Features	**Value**
E	Number of Roles, E_{role}	3
	Number of Rounds, E_{rounds}	100
	Number of connections per round, E_{length}	100
N	**Nodes**	
	Network devices or switches, N_{switch}	14
	Number of real client hosts, N_{real}	50
	Number of honey client hosts, N_{honey}	50
	Number of servers, N_{server}	6
L	**Connectivity**	
	Forwarding paths, $L_{src,dst}$	
	Maximum redundancy paths, $\|L_{src,dst}\|$	8
A	**Adversarial Features**	
	Compromised switches, A_{switch}	$\{1, 2\}$
	Target role, A_{role}	
	Attacker confidence on system, $A_{confidence}$	
P	**Set of Operational policy**	
	Connection definition, $P_{connection}$	
	Belief maintenance, P_{belief}	
	Attacker actions, $P_{attacker}$	

a transition the module can make by giving the variables new values; in the case of DTMCs, p_i is the transition probability. If the guard is true, each update is executed according to its probability. Modules interact with each other through synchronizing on identically labeled commands, thus modules can depend on each other for updates and transitions.

For modeling complex network behavior using PRISM, we developed a *code generator* that takes in a configuration and outputs PRISM models with necessary modules and transition formulas. The generated model also (1) ensures consistent state updates and module transitions; (2) identifies compromised switches based on observations from honey connections; and (3) generates the necessary reward functions to measure the performance of the system. Our framework generates a dedicated HoneyRoles model for each configuration. Mathematically, each configuration is defined by $\langle E, N, L, A, D, P\rangle$, as described in Table 5.1. We define three PRISM modules: *Defender*, *System* and *Adversary*. The interactions between these modules is summarized in Figure 5.2.

Defender Module

As shown in Figure 5.2, the defender module specifies the current system state by defining a connection $C \rightarrow \langle type, role, source, destination, path\rangle$. By selecting a new connection configuration, a new transition path is initiated. Listing 5.1 shows a simplified segment of PRISM code

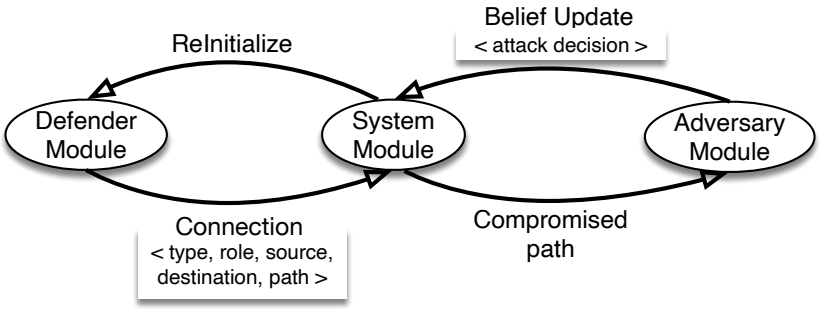


Figure 5.2: HoneyRoles workflow between modules in PRISM

for selecting a new connection configuration.

Both adversarial actions and the system belief update in the current path depend on the connection configuration. Since we cannot represent traffic replication in PRISM, we specify the same probabilistic selection weight for both the honey and real types. As a result, the model produces a nearly equal number of honey and real connections over the time.

For this implementation, we have only considered three mission-oriented roles, each of which is selected with equal probability. The source and destination are randomly chosen for each connection, depending on the type of connection and role chosen in previous states. For this PRISM analysis, we have considered both disjoint and non-disjoint paths. We are using a uniformly random distributed forwarding path selection algorithm. Since PRISM cannot directly encode a network topology, our PRISM code generator enumerates these different paths between sources and destinations as distinct PRISM formulas with unique tags.

System Module

The system module gets the current connection C as a configuration. It decides between two possibilities. If the chosen forwarding path contains at least one compromised switch, the system state gives control to the Adversary module. Otherwise, the system state moves towards the defender module to reinitialize the system.

Belief Update: After the adversary module takes actions (Section 5.5.1), control returns to the system module. For every round r, the system module records the number of honey connections (c_k) handled by each switch k, as well as the number of adversarial incidents (a_k). After the completion of each round c_k and a_k are reinitialized.

In our current implementation, each round consists of E_{length} connections (see Table 5.1). When completing one round, our model goes though approximately $E_{length} \times 20$ (or $\times$ 25) state transitions and $E_{length} \times 3$ module transitions. After completing a round, the current belief is calculated as described in Equation 5.1. Listing 5.2 shows a simplified segment of PRISM code representing this update. Here, if the current connection type is *honey* and *attack* is true, the

Listing 5.1: $P_{connection}$ code snippet

```
//#type selection#
[] sched=0 & free & flowType=0 -> (flowRatio/100): (flowType'=1) & (sched'=1)
  + (1-flowRatio/100): (flowType'=2) & (sched'=1);
//#role type selection #
[] sched=1 & free -> 1/3: (roleType'=role0) & (sched'=2) + 1/3: (roleType'=role1)
    & (sched'=2) + 1/3: (roleType'=role2) & (sched'=2);
//#source selection#
[] sched=2 & free & sourceID=0 & count<maxIteration & flowType=1 & roleType=role0 ->
  1/4: (sourceID'=h1) & (sourceSW'=h1sw) & (sched'= 3)+
  ... + 1/4: (sourceID'=h10) & (sourceSW'=h10sw) & (sched'= 3);
...
[] sched=2 & free & sourceID=0 & count<maxIteration & flowType=2 & roleType=role2 ->
  1/4: (sourceID'=h11) & (sourceSW'=h11sw) & (sched'= 3) +
  ... + 1/4: (sourceID'=h17) & (sourceSW'=h17sw) & (sched'= 3);
//# destination selection#
[] sched=3 & free & destinationID=0 & roleType=role0 -> 1/2: (destinationID'=h21) &
    (destinationSW'=h21sw)& (sched'= 4)
  + 1/2: (destinationID'=h24) & (destinationSW'=h24sw)& (sched'= 4);
...
//## current path selection ##
[] sched = 401 & sourceID != 0 & destinationID != 0 ->
    1/2 : (currentPath' = pathNumber0) & (sched'=402)
  + 1/2 : (currentPath' = pathNumber1) & (sched'=402);
```

adversarial incident count (a_k) of each switch k on the current *path* is incremented. Note that this code uses $\beta = 0.2$ as discussed in Section 5.5.2.

Adversary Module

To simulate reconnaissance, we assume the adversary receives all possible kinds (different types, roles, IDs) of connections from the defender that pass through the corresponding compromised switch. Note that we assume the adversary has knowledge observed from all compromised switches, if there is more than one. Ultimately, the goal of HoneyRoles is to defend against passive reconnaissance by deceiving the adversary with honey flows. In the context of our model, this means the defender cannot detect the adversary or update its belief. However, an adversary that is aware of HoneyRoles may still act, performing some active reconnaissance and attacks once it has gained sufficient confidence through passive reconnaissance. Therefore, the adversary module accumulates confidence in observed information and then attempts to (1) increase confidence through some active reconnaissance, and (2) attack real connections with targeted roles.

As specified in Table 5.1, the adversary module has a target organizational role A_{role} (e.g., IT administrators). We assume the adversary is only interested in traffic for that role, as defined by connections to the role's corresponding target servers. The module is also configured with a confidence parameter $A_{confidence}$, which specifies a threshold of sufficient belief in observed information.

The adversary module has two phases: (1) attack, and (2) build confidence. For the at-

Listing 5.2: $R_{k,t}$ update code snippet from P_{belief}

```
// #record honey events#
[beliefUpdateAttacker] sched=5 & active_defender=true &
  (attackFlow=2| attackFlow=3) & (flowType=2) ->
  (sched'=6) & (received_size'=0);
...
[] sched=6 & currentPath=3002 -> (ae_sw0'=ae_sw0+1) &
  (ae_sw4'=ae_sw4+1) & (ae_sw1'=ae_sw1+1) & (sched'=801);
...
[] (flowType=2) & sched=8 & currentPath=3002 ->
  (count_sw0'=count_sw0+1) & (count_sw4'=count_sw4+1) &
  (count_sw1'=count_sw1+1) & (sched'=(80001));
...
// #update belief#
[] sched=9 & roundFinished=true ->
  (bi_sw0'=round((((bi_sw0/100)*0.80) +
  ((ae_sw0/(count_sw0))*0.20))*100)) &
  ... & (bi_sw7'=round((((bi_sw7/100)*0.80)+
  ((ae_sw7/(count_sw7))*0.20))*100)) & (sched'=10);
[] sched=10 & roundFinished=true ->
  (count_sw0'=1) & (ae_sw0'=1) & ... & (sched'=11);
```

tack phase, each connection starts with checking whether the current connection as associated with A_{role}. If the current connection matches A_{role}, the adversary probabilistically (based on $A_{confidence}$) determines its belief for the current observation. If the adversary believes the current observation is real traffic, it performs an attack. On the other hand, if the adversary believes the current observation is honey traffic, it does nothing. Listing 5.3 shows a simplified segment of PRISM code for the adversary module.

On the completion of each round, the adversary probabilistically updates $A_{confidence}$, either increasing or decreasing it. To indicate that the adversary's knowledge is increasing with each connection, our implementation uses a higher weight (e.g., $\frac{2}{3}$ in Listing 5.3) for increasing the $A_{confidence}$. To simulate the effect of deception, we also include the possibility of decreasing confidence (e.g., $\frac{1}{3}$ in Listing 5.3). Finally, we assume the adversary cannot have 100% confidence over its observation. Therefore, if $A_{confidence} \leq 90$, our implementation uses an equal probability of increasing or decreasing $A_{confidence}$ (e.g., $\frac{1}{2}$ in Listing 5.3). Alternatively, if $A_{confidence} \geq 90$, our implementation only allow it to remain same or decrease.

Based on this operation, the adversary's action for a connection can be defined by a Markov chain. Let HoneyRoles's initial state be denoted $\langle C_0, A_0, B_0 \rangle$, where C_0 is the current connection state, A_0 is the adversary state in terms of confidence, and B_0 indicates system's belief on the suspiciousness of switches. Figure 5.3 provides a simplified visualization. The figure assumes only three possible conditions: (1) *compromised* defines the state of a forwarding path being compromised or not, (2) *type* defines a connection to be either honey or real, and (3) *attack* defines an adversarial attack decision. We assume a connection configuration (e.g., source, destination, type) can repeat; however, this is infrequent and not shown in the figure.

Listing 5.3: $P_{attacker}$ code snippet

```
confidence: int init 10;
[startAttacker] attacker=0 & active_attacker=true & sched=5 -> (attacker'=1);
[] attacker=1 & active_attacker=true & targetRole=roleType ->
  (confidence/100): (beliefObservation'=true) & (attacker'=2)
  +(1-confidence/100): (beliefObservation'=false) & (attacker'=2);
[] attacker=2 & active_attacker=true & beliefObservation=true & flowType=1->
(attack'=true) & (attacker'=3);
[] attacker=2 & active_attacker=true & beliefObservation=false & flowType=2->
(attack'=true) & (attacker'=3);
[] attacker=2 & active_attacker=true & beliefObservation=false & flowType=1->
(attack'=false) & (attacker'=3);
[] attacker=2 & active_attacker=true & beliefObservation=true & flowType=2->
(attack'=false) & (attacker'=3);
...
[] attacker=6 & attackerRoundComplete=true & confidence<90 ->
  2/3: (confidence'=confidence+1) & (attacker'=7)
  + 1/3: (confidence'=confidence-1) & (attacker'=7);
[] attacker=6 & attackerRoundComplete=true & confidence>=90 ->
  1/2: (attacker'=7)+ 1/2: (confidence'=confidence-1) & (attacker'=7);
[beliefUpdateAttacker] (attacker=7|attacker=6) & attackFlow!=0 &
  active_defender=true & active_attacker=false -> (attacker'=0);
```

5.5.2 Security Evaluation

This section provides the simulation results from the PRISM module described in Section 5.5.1. However, first we describe our experimental setup and performance metrics.

Experimental Setup: As described in Section 5.5.1, our code generator automatically creates a PRISM model given a system configuration. The code generator was written in around 1,300 lines of Python code. It has two parts: 1) the *TopologyParser* generates the topology by using connectivity information to enumerate all-possible forwarding paths for each pair of edge switches, 2) the *PRISMCodeGenerator* takes the topology information and the system parameters (Table 5.1) and generates final PRISM logic. The "Experiment" column in Table 5.1 specifies the configuration used for our experiment. Specifically, we considered scenarios where there were 1 or 2 compromised switches, including simulations where the compromised switch resided at different locations within the Fat-Tree topology (i.e., edge, aggregate, core). Note that we used a Fat-Tree topology for lack of a public database of an enterprise network topology. Repositories such as Topology-Zoo [118] and Internet2 [28] only include topologies for data centers, ISPs, and point of presence (POP) networks. However, our code generator can consume topologies in Geography Markup Language (GML) following the format of Topology-Zoo and can therefore be easily used to evaluate different topologies. Finally, to assess the sensitivity of β in Equation 5.1, we ran the simulator with $\beta \in \{0.1, 0.2, 0.3, 0.4, 0.5\}$.

We used the discrete-event simulator built into PRISM, a technique often called statistical model checking [3]. This sampling approach generates a large number of random paths through the model, evaluating the result of the given properties on each run, and using this information

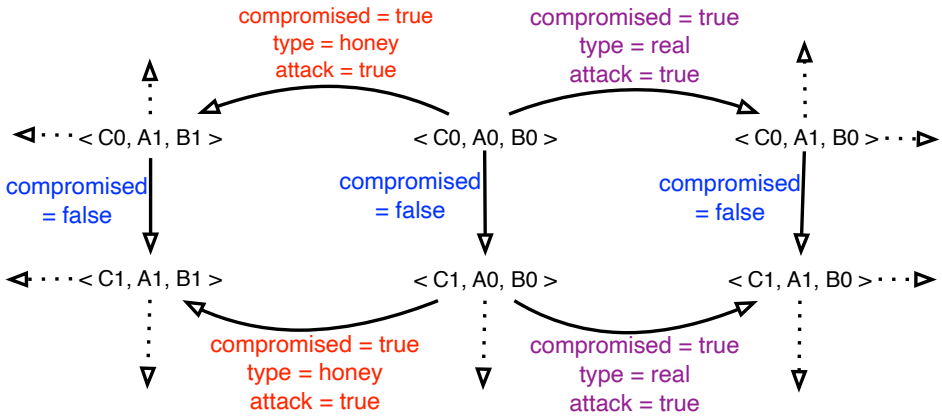
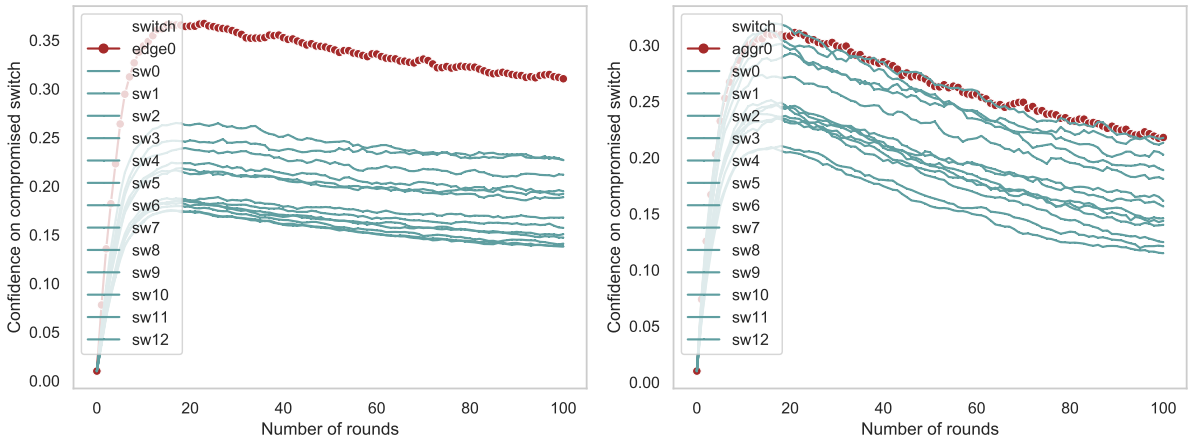
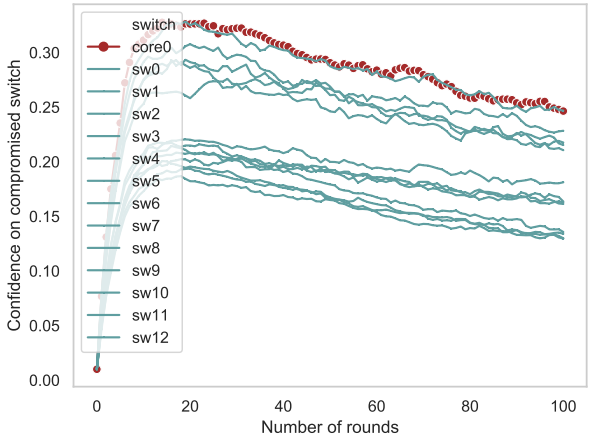


Figure 5.3: A simplified version of HoneyRoles Markov chain



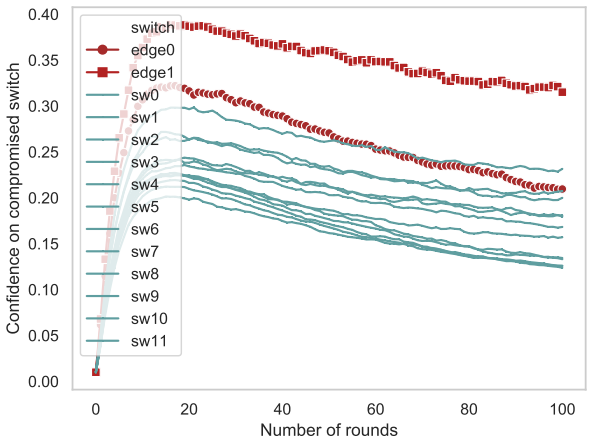
(a) One compromised edge switch

(b) One compromised aggregate switch

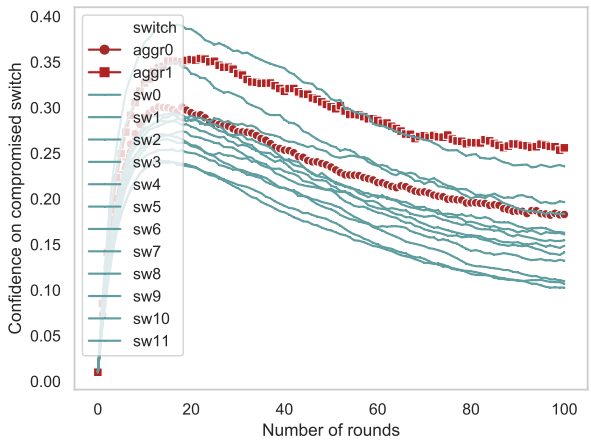


(c) One compromised core switch

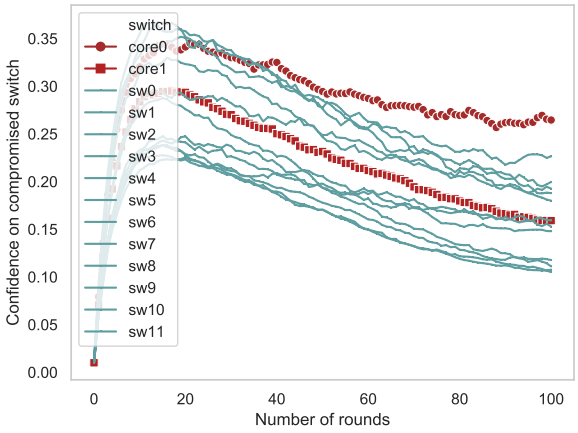
Figure 5.4: Confidence in switch compromise for one compromised switch ($\beta = 0.2$).



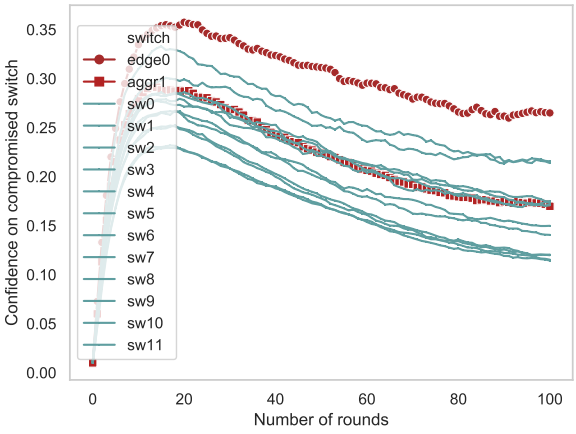
(a) Two compromised edge switches



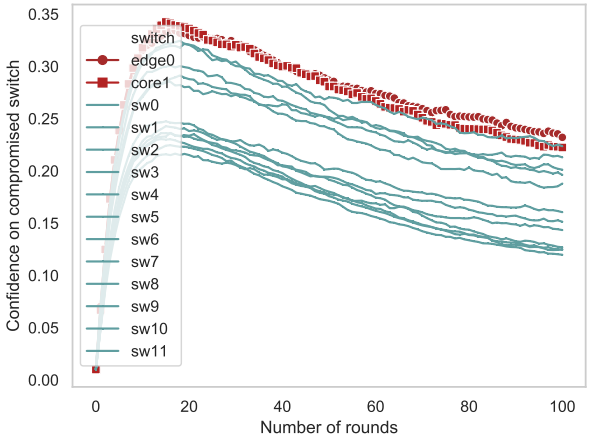
(b) Two compromised aggregate switches



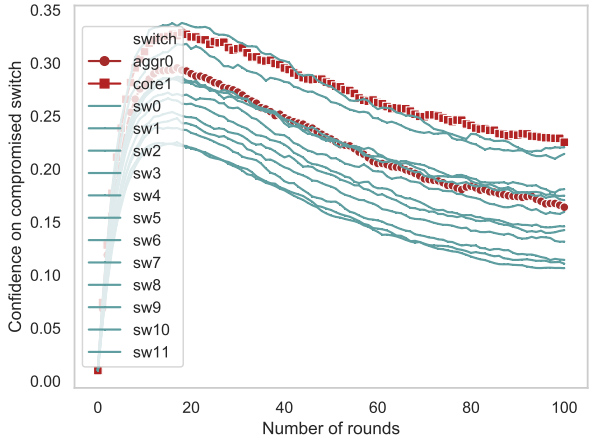
(c) Two compromised core switches



(d) One edge and one aggregate compromised switch



(e) One edge and one core compromised switch



(f) One aggregate and one core compromised switch

Figure 5.5: Confidence in switch compromise for two compromised switches ($\beta = 0.2$).

to generate an approximately correct result [89]. Each simulation was run for the path length of $200,000$ (approximately 100 rounds) and collected data on each step of $2,000$ (approximately on completion of each round). Each simulation takes 50 samples and provides the mean values as a final result.

Performance Metrics: Recall that this evaluation is designed to determine how well HoneyRoles can locate the compromised switch or switches, i.e., how often the compromised switch(es) appear high in HoneyRoles' suspiciousness ranking. We thus examine this ranking over the course of 100 rounds.

Detection Accuracy with One Compromised Switch

Figure 5.4 shows the relative ranking of suspiciousness for switches for $\beta = 0.2$ when there is only one compromised switch. The other β configurations produced anecdotally similar graphs, but as hypothesized, a smaller β performs better. The figure shows that when there is one compromised switch, that switch is consistently ranked in the top-1 or top-2.

When comparing the different locations for the compromised switch (i.e., edge, aggregate, and core), the figure shows the best performance for compromised switches located at the edge (Figure 5.4a). This is because it is easier to isolate the attack activity over the time. As shown in Figures 5.4b and 5.4c, when a core or aggregate switch is compromised, HoneyRoles does not provide as clear of a distinction. However, this is an artifact of the Fat-Tree topology, as core and aggregate switches are included in most of the forwarding paths that raise alarms. Hence, it is difficult to statistically determine which switch on the path is performing the attacks. That said, even with this high overlap, the compromised switches were within the top-2 riskiest at all times.

Detection Accuracy with Two Compromised Switches

Figure 5.5 shows six possible combinations of compromised switches for $\beta = 0.2$. Other than the number of compromised switches, the other parameters remained the same as in the tests with one compromised switch. As before, different β values produced visually similar results, with smaller β values performing better.

As discussed in Section 5.5.2, edge switches are easier to isolate than aggregate and core. When aggregate or core switches are compromised, at least one of the compromised switches is ranked in the top one or two most of the time, with the second compromised switch being in the top five for all but two scenarios. Note that the network administrator can approach refreshing switches to a good known state in an incremental fashion. That is, it can refresh the top-1 switch, removing one of the compromised switches and leaving only one, which as shown in Figure 5.4 is easier to isolate. While the adversary will know that it has been detected, in the worst case (for detection) it will stop attacking connections, which is ultimately our goal. The

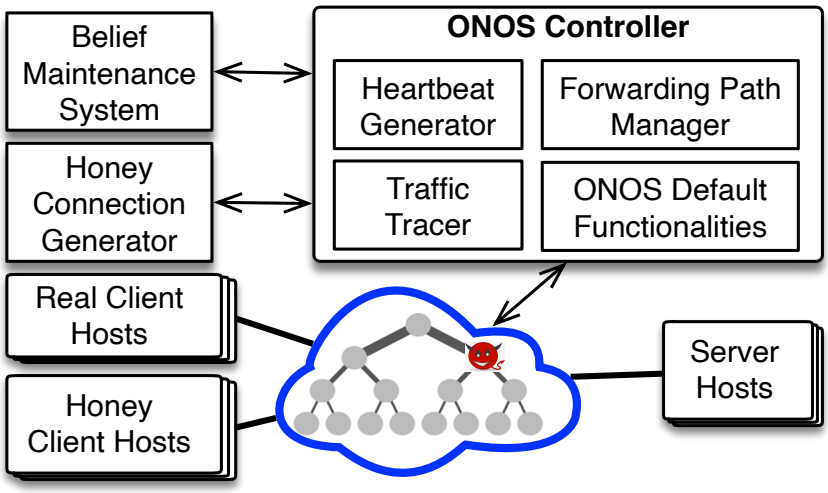


Figure 5.6: Experimental Layout for Evaluation

system administrators can also define some threshold on the switch risk factors, depending on their security requirement. Thus, administrators will remove a switch only when its risk factor goes beyond that threshold.

5.6 Performance Evaluation

HoneyRoles's security stems from its deception elements (e.g., honey connections and routes), which add network overhead. We now discuss our prototype implementation, experimental setup, and evaluate HoneyRoles's performance overhead in an emulated Mininet [140] environment.

5.6.1 Implementation

Our HoneyRoles prototype is implemented as six components that comprise the design in Section 5.4. We built our prototype on top of the OpenJDK 11.0.7 and ONOS 2.0.0 SDN controller with the default configuration. Three components are implemented as ONOS Java applications: ForwardingPath Manager (95 lines of code), Heartbeat Generator (305 lines of code), and Traffic Tracer (250 lines of code). Two additional components run as dedicated processes that communicate with the ONOS controller: Belief Management System (180 lines of code) and Honey Connection Processor (310 lines of code). The Mininet network creation and real host traffic generator took up 600 and 120 lines of code, respectively. The Honey Agent (240 lines of code) is used implementation the workflow of a honey host. Although HoneyRoles can function with different applications (e.g., SSH, SMTP), we restricted ourselves to HTTP/HTTPS traffic.

Network Creation: Our performance analysis uses the same Fat-Tree topology generation algorithm used for the security analysis in Section 5.5. Both *Real* and *Honey* client hosts are implemented as standard Mininet hosts. The Real hosts execute a script that randomly initiates sessions (a sequence of one or more HTTP requests) with a target server. The Honey hosts execute the Honey Agent script, which uses *heartbeat* instructions from the controller to initiate a session with a target server and then reports results back to the controller using *Honey Notifications*. Servers are implemented as Docker containers running on the host machine.

5.6.2 Experimental Setup

The evaluation was hosted in a virtual machine configured with 8 vCPUs and 32 GB RAM, running on a VMware ESXi 6.5.0 host with Intel(R) Xeon(R) CPU E5620 @ 2.40GHz processors. Figure 5.6 shows the network, along with the main components described in the implementation details (Section 5.6.1). We considered four environments to compare the performance impact of various HoneyRoles components. The *Baseline* environment was configured to use the default ONOS settings with no HoneyRoles features enabled and included only real hosts in the network. The *Honey Forwarding* environment replaces the default ONOS reactive forwarding application with the HoneyRoles Forwarding application but does not introduce honey hosts or honey agents. The *Honey Host* environment was configured with the default ONOS forwarding app but introduces the Heartbeat Generation Application and honey hosts. Here, we have one honey host initiated in correspond to each real host in the network. Finally, the HoneyRoles environment was configured with all HoneyRoles features enabled and one honey host for each real host in the network.

To maintain consistency with our security analysis, we configured each environment with 50 real hosts. As discussed easier in Section 5.4, we are using a 5-tuple for flow rule matching: s_{ip}, s_{port}, d_{ip}, d_{port}, and $protocol$. The baseline and all treatments use ONOS's default 10 second idle timeout for flow-mod rules. Each experiment ran for 30 minutes and all hosts (honey or real) were configured to send 1 request per second to a specific server, which is selected based on their roles from a fixed set.

5.6.3 HoneyRoles Performance Overhead

Calculating a single average overhead across all pairs does not provide a useful characterization, as different pairs have different numbers of hops between them, resulting in a significant variance in completion time. Therefore, to observe the overhead HoneyRoles imposes on real network traffic, we calculated the average request completion time between each unique real-client server pair for the baseline environment. For each non-baseline environment, we calculated the percent overhead of every real request from the baseline average. We plotted the percent overheads for each configuration as a cumulative distribution function (CDF).

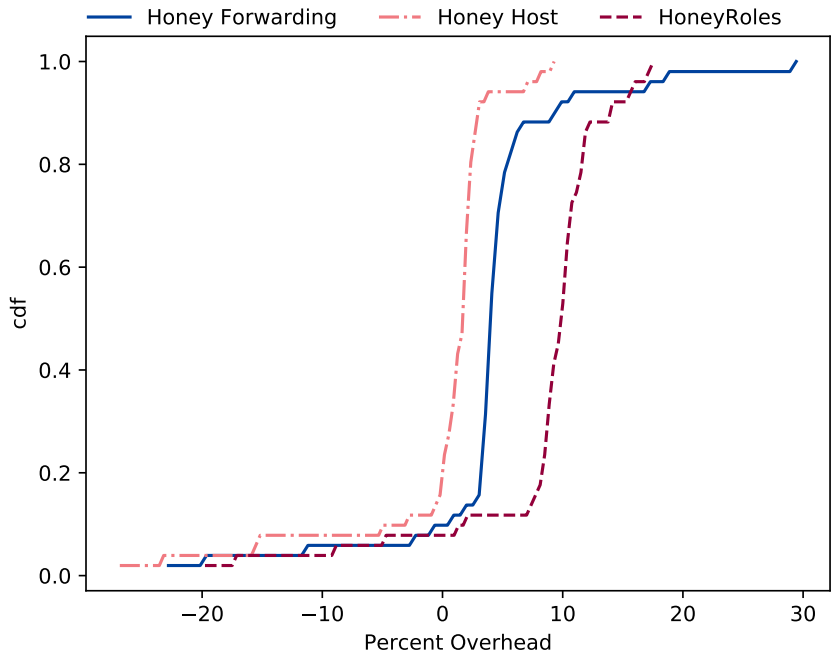


Figure 5.7: Percent Overhead of HTTP request completion time in each configuration compared to baseline.

Figure 5.7 depicts the overhead of each treatment with respect to the baseline. Each line represents the percentage of real requests in each environment that finished under a given percent overhead calculated using the average baseline request completion time for unique client-server pairs. That is, each request between client c and server s in the Honey Forwarding Application, Honey Host, and HoneyRoles environments was compared to the average completion time of all requests between c and s in the baseline environment. From this graph we observe that for HoneyRoles, 90% of requests finish with less than 14% overhead when compared to the baseline.

Note that these percentage overheads are for small request-completion times, which are significantly impacted by jitter. The median request completion time in the baseline environment was 31 ms. As a result, even small changes in completion time in the other environments show as a larger magnitude overhead. For example, with a 31 ms baseline completion time, a request with a 9 ms increase from the baseline (e.g., 40 ms) results in a 29% overhead. The natural jitter in network requests and the sensitivity when dealing with small numbers can also explain the negative overheads observed in Figure 5.7.

Further, we compared the average request-completion time of each environment to the baseline average by calculating the effect size using *Cohen's d*. Cohen's d reports how many pooled standard deviations two groups differ by. According to Cohen [35], a d value of 0.20 is considered a "small" effect, a d value of 0.50 is a "medium" effect, and a d value of 0.80 is a

"large" effect of an experimental change to a control group. For the HoneyRoles environment, we observe 55% of client-server pairs have a d value of under 0.20 (small effect), and all of d values are below 0.63, which is well below the 0.80 margin (large effect). Thus we observe that, with respect to request-completion times, HoneyRoles had a small effect for a majority of the client-server pairs and a medium effect for all the rest of the pairs.

We believe our small overheads are due to two primary reasons. First, the network is not under full load, thus the introduction of Honey Host traffic does not compete with real traffic for resources and has minimal impact on the network links and server processing. Second, although the Honey Forwarding application may select non-optimal routes for traffic, for the choice of Fat-Tree the non-optimal routes do not introduce major differences in request completion time. It may be possible that a network is designed in such a way that a non-optimal route may introduce much higher round trip times but these routes are not permanent and some traffic will still travel over optimal or close to optimal routes. Essentially, network administrators can create additional network links to provide shorter alternative paths.

5.7 Discussion

Attack variations: HoneyRoles considers that an adversary uses *passive* and *active* reconnaissance to obtain knowledge about target enterprise roles, presumably to launch active attacks using that knowledge. Many active attacks and reconnaissance techniques have been discovered over the past decades. We envision a HoneyRoles deployment will include a collection of attacks (e.g., SSL downgrade, wrong SSL certificate, page contents modified) and detection types (e.g., packet rerouting, packet hijacking, manipulation) for different types of applications (e.g., SMTP, FTP). However, the SSL-stripping and blackholing detectors are sufficient to demonstrate the heartbeats and reports functions, because they cover the spectrum of modification and dropping.

Accuracy vs. deception: Using the centralized control of SDN, it is possible to dynamically change honey components according to system belief to improve accuracy. However, such an approach would be risky. First, sudden changes in system behavior may alarm the adversary and reduce the effectiveness of the deception (e.g., helping it identify which IP address belong to honey hosts). Second, a dynamic change in system behavior may increase complexity in large networks. Third, the TCB must include at least a segment of switches to achieve the security goal.

Scope of implementation: We used PRISM to evaluate the security of HoneyRoles and used an emulated Mininet environment to measure performance overhead. These evaluation frameworks are approximations of realistic enterprise networks. Our security evaluation was limited in the way it modeled attacker behavior, as we could not find any realistic attack data for enterprise reconnaissance. Absent realistic attack behavior, the PRISM model was more

comprehensive than a Mininet simulation to estimate detection accuracy. Additionally, we were unable to find realistic enterprise network topologies and relied on the Fat-Tree topology as a representative topology with redundant links and switches. Finally, as stated in Section 5.3, we assume the existence of an ambient network traffic generator [146, 135, 22], which our implementation does not include. Additional work is required to design and evaluate ambient network traffic generators against more recent machine learning algorithms; however, doing so is orthogonal to the contributions of this work. We also note that some machine learning algorithms require significant storage and computational capabilities, which are not available to an adversary positioned on a compromised packet forwarding device.

5.8 Summary

The increasing complexity of packet forwarding devices such as routers and switches make them a new target for advanced persistent threats. From the vantage point of a compromised packet forwarding device, an adversary can passively monitor network traffic to identify not only the network topology and servers listening on ports, but also the client hosts that connect to high-value servers such as domain controllers and financial systems. In this chapter, we presented HoneyRoles as a novel approach to defending against this relatively new threat. HoneyRoles uses honey connections to both deceive adversaries and dissuade them from performing attacks. A key idea behind HoneyRoles is to focus on client hosts performing high-value organizational roles, building metaphorical haystacks around their network traffic. The honey connections used to build these haystacks also act as network canaries to bait adversaries and more quickly detect their presence. We built a prototype of HoneyRoles in an SDN environment and modeled its operation using the PRISM probabilistic model checker. In doing so, we found that HoneyRoles reliably ranks compromised switches among the most suspicious while having only a small effect on network request completion time. As such, we believe role-based network deception is a promising approach for defending against adversaries that have compromised network devices.

Chapter 6

Conclusion and Future Directions

This dissertation explored role-based access control and deception using 5-tuple network information to defend against insider threats in an enterprise network environment. First, we concentrated on the problem of compromised hosts. In Chapter 3, we led an effort to define least-privilege network access control where each host has a different, limited view of the other hosts and services within a single-site enterprise network. The main contribution of NetViews was to design a policy framework that can capture the semantic gap between networking primitives and an enterprise's organizational structure. In Chapter 4, we demonstrated MultiSite NetViews (MSNetViews), which extends the single, globally-defined NetViews to many geographically distributed sites. We incorporated centralized policy management to maintain administrative access control, policy-update correctness, and *need-to-know* policy distribution across the sites. We achieved resiliency and performance using distributed policy enforcement. Finally, we addressed the problem of compromised forwarding devices in Chapter 5 through a novel deceptive defense approach against passive and active reconnaissance. HoneyRoles coordinates deceptive honey connections by modeling fake hosts organized into roles corresponding to client hosts' organizational functions and real-time traffic patterns. The honey connections act as network canaries also; to bait adversaries and isolate their presence.

Furthermore, the implementation and evaluation of these works demonstrates the utility and effectiveness of reactive forwarding in defending against insider threats. We designed and implemented these systems using reactive Software Defined Networks (SDN). We used ONOS as the controller, where ONOS's Intent framework enabled efficient forwarding rule management, and ONOS's reactive forwarding framework enabled dynamic path selection. Our performance analysis showed that the latency and throughputs are comparable to baseline reactive forwarding using a Mininet-based emulated environment. In NetViews and MSNetViews, we leveraged NIST's Next Generation Access Control (NGAC) for policy definition and management. Our security analysis proved the effectiveness against insider attackers in NetViews and the restrictiveness of the policy definition in MSNetViews. In HoneyRoles, we did a probabilistic security

analysis using the *PRISM probabilistic model checker* and proved the reliability of the isolation mechanism. As such, we established that the role-based access control and network deception using reactive SDN as an effective approach in achieving zero trust within on-premises enterprise networks.

Motivated by the findings in this dissertation, there are several future directions for research, including:

Access control frameworks in containerized environments In the cloud environment, there is not only traffic traveling from service to user (north-south) but also service-to-service (east-west). All the communication among microservices is initiated and carried out with network calls in a containerized environment. This "east-west" traffic can also contain confidential data, and exposing this communication among microservices creates potential attack surfaces (e.g., man-in-the-middle attacks, reconnaissance). Thus this leads us to the question, *How to implement Zero Trust for service-to-service communication in the containerized environment?*.

In recent times, the Kubernetes and Istio [54] service mesh has enabled the least privilege policy enforcement, and Cilium [71] can provide deep traffic visibility; Hence, there is a potential of achieving Zero Trust in the containerized environment. However, its yet to explored formally whether the existing system components provide the least privileged access control and deep traffic visibility. Furthermore, each cloud service provider has its own way of implementing services and access control. On the other hand, there is a syntactic and semantic gap among the different policies and service-to-service dependency. We believe, understanding and analyzing different implementations and techniques of east-west traffic coordination from the zero-trust point of view will allow us to address these challenges.

Access control in multi-cloud environment Multi-cloud environments are becoming more popular; ninety-four percent of organizations use not just one but multiple cloud environments to support their business [52]. The reason behind these multi-cloud provisions can be avoiding vendor lock-in, requiring different services from different providers, or cloud bursting (a workload in a private cloud bursts into a public cloud when the need arises). Multi-cloud environments exacerbate the research challenges of defining zero-trust communication in and among cloud-based services (CSPs); these CSPs have different implementation models. Furthermore, understanding how access control policies from multiple regulatory bodies (e.g., enterprise, government, cloud provider) are maintained is vital for transparency and zero-trust security.

Least privilege for Cyber Physical Systems (CPS) Cyber-Physical Systems (CPS) typically involve various interconnected systems (e.g., sensors and actuators) that monitor and manipulate real objects and processes. These systems empower our critical infrastructure; they

form the basis for emerging and future smart services. Due to their heterogeneous nature, reliance on private and sensitive data, and large-scale deployment, the attack surface is also broadening. CPS environments are highly contextualized and dynamic. Devices have resource constraints in terms of battery and computational power. These characteristics make it harder to support security functions on devices, which means the network is the primary security layer. On top of that, sensor networks can often be off-network or follow different standards for wireless communication. Hence, incorporating NetViews-like defense in a network of CPS devices with a unique threat model is challenging. Research efforts exploring different system designs, attack scenarios, and implementations will allow CPS environments to achieve zero trust security.

REFERENCES

[1] Stefan Achleitner, Thomas La Porta, Patrick McDaniel, Shridatt Sugrim, Srikanth V. Krishnamurthy, and Ritu Chadha. Cyber deception: Virtual networks to defend insider reconnaissance. In *Proceedings of the 8th ACM CCS International Workshop on Managing Insider Security Threats*, pages 57–68, 2016.

[2] R. A. Addad, D. L. C. Dutra, M. Bagaa, T. Taleb, H. Flinck, and M. Namane. Benchmarking the ONOS Intent Interfaces to Ease 5G Service Management. In *Proceedings of the IEEE Global Communications Conference*, GLOBECOM, 2018.

[3] Gul Agha and Karl Palmskog. A survey of statistical model checking. *ACM Trans. Model. Comput. Simul.*, 28(1), January 2018.

[4] Mohammed H Almeshekah and Eugene H Spafford. Cyber security deception. In *Cyber deception*, pages 23–50. 2016.

[5] Rashid Amin, Martin Reisslein, and Nadir Shah. Hybrid sdn networks: A survey of existing approaches. *IEEE Communications Surveys & Tutorials*, 20(4), 2018.

[6] Blake Anderson and David McGrew. Tls beyond the browser: Combining end host and network data to understand application behavior. In *Proceedings of the Internet Measurement Conference*, page 379–392, New York, NY, USA, 2019. Association for Computing Machinery.

[7] J. P. Anderson. Computer Security Technology Planning Study. Technical report, Air Force Electronic Systems Division, 1972.

[8] Iffat Anjum. Single Site Netviews. GitHub, 2021. `https://github.com/netviews/ss-netviews`.

[9] Iffat Anjum, Daniel Kostecki, Ethan Leba, Jessica Sokal, Rajit Bharambe, William Enck, Cristina Nita-Rotaru, and Bradley Reaves. Removing the reliance on perimeters for security using network views. In *Proceedings of the ACM Symposium on Access Control Models and Technologies*, SACMAT, 2022.

[10] Iffat Anjum, Mohammad Sujan Miah, Mu Zhu, Nazia Sharmin, Christopher Kiekintveld, William Enck, and Munindar P Singh. Optimizing vulnerability-driven honey traffic using game theory, 2020.

[11] Markku Antikainen, Tuomas Aura, and Mikko Särelä. Spook in your network: Attacking an sdn with a compromised openflow switch. In Karin Bernsmed and Simone Fischer-Hübner, editors, *Secure IT Systems*. Springer International Publishing, 2014.

[12] S. Antonatos, P. Akritidis, E.P. Markatos, and K.G. Anagnostakis. Defending against hitlist worms using network address space randomization. *Computer Networks*, 51(12), 2007.

[13] MITRE ATT&CK. NotPetya, 2019. `https://attack.mitre.org/software/S0368/`.

[14] Michael Backes, Goran Doychev, and Boris Köpf. Preventing side-channel leaks in web traffic: A formal approach. In *20th ISOC Network and Distributed System Security Symposium*, 02 2013.

[15] Genevieve Bartlett, John Heidemann, and Christos Papadopoulos. Understanding passive and active service discovery. In *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, pages 57–70, 2007.

[16] R. Basnet, S. Mukherjee, V. M. Pagadala, and I. Ray. An efficient implementation of next generation access control for the mobile health cloud. In *Proceedings of the International Conference on Fog and Mobile Edge Computing*, FMEC, 2018.

[17] Noam Ben-Asher and Cleotilde Gonzalez. Effects of cyber security knowledge on attack detection. *Computers in Human Behavior*, 48, 2015.

[18] Kevin Benton, L. Jean Camp, and Chris Small. Openflow vulnerability assessment. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN '13, pages 151–152. ACM, 2013.

[19] Chafika Benzaïd, Tarik Taleb, and Muhammad Zubair Farooqi. Trust in 5g and beyond networks. *IEEE Network*, 35(3), 2021.

[20] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and David Walker. P4: Programming Protocol-independent Packet Processors. *ACM SIGCOMM Computer Communication Review*, 44(3), July 2014.

[21] Brian M. Bowen, Vasileios P. Kemerlis, Pratap Prabhu, Angelos D. Keromytis, and Salvatore J. Stolfo. A system for generating and injecting indistinguishable network decoys. *J. Comput. Secur.*, 20(2–3):199–221, March 2012.

[22] Brian M. Bowen, Vasileios P. Kemerlis, Pratap Prabhu, Angelos D. Keromytis, and Salvatore J. Stolfo. A system for generating and injecting indistinguishable network decoys. *J. Comput. Secur.*, 20(2-3):199–221, 2012.

[23] William Brockelsby and Rudra Dutta. Traffic Analysis in Support of Hybrid SDN Campus Architectures for Enhanced Cybersecurity. In *Proceedings of the Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN)*, March 2021.

[24] BitWizard B.V. Mtr. BitWizard, 1997. `http://www.bitwizard.nl/mtr/`.

[25] Claudio Calcaterra, Alessio Carmenini, Andrea Marotta, and Dajana Cassiol. Hadoop performance evaluation in software defined data center networks. In *Proceedings of the International Conference on Computing, Networking and Communications*, ICNC, 2019.

[26] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. Gude, N. McKeown, and S. Shenker. Rethinking enterprise network control. *IEEE/ACM Transactions on Networking*, 17:1270–1283, Aug 2009.

[27] Martin Casado, Michael J. Freedman, Justin Pettit, Jianying Luo, Nick McKeown, and Scott Shenker. Ethane: Taking Control of the Enterprise. In *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM, 2007.

[28] Internet2 Network Operations Center. Internet2, 1996.

[29] Ping Chen, Lieven Desmet, and Christophe Huygens. A study on advanced persistent threats. In *IFIP International Conference on Communications and Multimedia Security*, pages 63–72. Springer, 2014.

[30] William R. Cheswick, Steven M. Bellovin, and Aviel D. Rubin. *Firewalls and Internet Security: Repelling the Wily Hacker*. Addison-Wesley Professional, 2003.

[31] Po-Wen Chi, Chien-Ting Kuo, Jing-Wei Guo, and Chin-Laung Lei. How to detect a compromised SDN switch. In *Proceedings of the 1st IEEE Conference on Network Softwarization (NetSoft)*, pages 1–6, April 2015.

[32] Catalin Cimpanu. Cisco bungled RV320/RV325 patches, routers still exposed to hacks. ZDNet, March 2019. `https://www.zdnet.com/article/cisco-bungled-rv320rv325-patches-routers-still-exposed-to-hacks/`.

[33] Department of Homehald Security CISA. Petya Ransomware. Alert (TA17-181A), 2017. `https://us-cert.cisa.gov/ncas/alerts/TA17-181A`.

[34] Andrew Clark, Kun Sun, and Radha Poovendran. Effectiveness of ip address randomization in decoy-based moving target defense. In *52nd IEEE Conference on Decision and Control*, pages 678–685, 12 2013.

[35] Jacob Cohen. *Statistical Power Analysis for the Behavioral Sciences*. Routledge Member of the Taylor and Francis Group, second edition, 1988.

[36] Daniel Conde. Software-defined perimeters: An architectural view of sdp. `https://sdn.ieee.org/newsletter/march-2017/software-defined-perimeters-an-architectural-view-of-sdp`, 2017.

[37] Mauro Conti, Fabio De Gaspari, and Luigi V. Mancini. Know your enemy: Stealth configuration-information gathering in sdn. *Lecture Notes in Computer Science*, page 386–401, 2017.

[38] E. Coyne and T. R. Weil. ABAC and RBAC: Scalable, Flexible, and Auditable Access Management. *IT Professional*, 15(03), May 2013.

[39] Mohan Dhawan, Rishabh Poddar, Kshiteej Mahajan, and Vijay Mann. Sphinx: Detecting security attacks in software-defined networks. In *ISOC Network and Distributed System Security Symposium*, 2015.

[40] Jon Dugan, Seth Elliott, Bruce A. Mah, Jeff Poskanzer, and Kaustubh Prabhu. iPerf - The ultimate speed test tool for TCP, UDP and SCTP, 2015. `https://iperf.fr/`.

[41] R. J. Enbody and A. K. Sood. Targeted cyberattacks: A superset of advanced persistent threats. *IEEE Security & Privacy*, 11:54–61, 2013.

[42] Doug Barth Evan Gilman. *Zero Trust Networks*. Building Secure Systems in Untrusted Networks. O'Reilly Media, Inc., July 2017.

[43] Seyed Kaveh Fayazbakhsh, Luis Chiang, Vyas Sekar, Minlan Yu, and Jeffrey C. Mogul. Enforcing Network-Wide Policies in the Presence of Dynamic Middlebox Actions using FlowTags. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation*, NSDI, 2014.

[44] Andrew D. Ferguson, Steve Gribble, Chi-Yao Hong, Charles Killian, Waqar Mohsin, Henrik Muehe, Joon Ong, Leon Poutievski, Arjun Singh, Lorenzo Vicisano, Richard Alimi, Shawn Shuoshuo Chen, Mike Conley, Subhasree Mandal, Karthik Nagaraj, Kondapa Naidu Bollineni, Amr Sabaa, Shidong Zhang, Min Zhu, and Amin Vahdat. Orion: Google's Software-Defined networking control plane. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, pages 83–98, 2021.

[45] David Ferraiolo. Unpacking Next Generation Access Control (NGAC) and Tetrate Q. TETRATE, 2019. `https://www.tetrate.io/blog/unpacking-next-generation-access-control-ngac-and-tetrate-q/`.

[46] David Ferraiolo, Vijayalakshmi Atluri, and Serban Gavrila. The Policy Machine: A novel architecture and framework for access control policy specification and enforcement. *JOURNAL of Systems Architecture*, 57(4), 2011.

[47] David Ferraiolo, Ramaswamy Chandramouli, Rick Kuhn, and Vincent Hu. Extensible Access Control Markup Language (XACML) and Next Generation Access Control (NGAC). In *Proceedings of the ACM International Workshop on Attribute Based Access Control*, ABAC, 2016.

[48] David F Ferraiolo, Larry Feldman, and Gregory A Witte. Exploring the next generation of access control methodologies. NIST, 2016. `https://www.nist.gov/publications/exploring-next-generation-access-control-methodologies`.

[49] FireEye. Highly Evasive Attacker Leverages SolarWinds Supply Chain to Compromise Multiple Global Victims With SUNBURST Backdoor. THREAT RESEARCH, 2020. `https://www.fireeye.com/blog/threat-research/2020/12/evasive-attacker-leverages-solarwinds-supply-chain-compromises-with/sunburst-backdoor.html`.

[50] Project Floodlight. Floodlight. `http://www.projectfloodlight.org/floodlight/`, 2019.

[51] Rodrigo Fonseca, George Porter, Randy H. Katz, Scott Shenker, and Ion Stoica. X-trace: A pervasive network tracing framework. In *Proceedings of the 4th USENIX Conference on Networked Systems Design & Implementation*, NSDI'07, pages 20–20. USENIX Association, 2007.

[52] Forbes. 90% of companies have a multicloud destiny: Can conventional analytics keep up?, 2021-09-09. `https://www.forbes.com/sites/googlecloud/2022/03/04/90-of-companies-have-a-multicloud-destiny-can/conventional-analytics-keep-up/?sh=552d07465d89`.

[53] Nate Foster, Rob Harrison, Michael J. Freedman, Christopher Monsanto, Jennifer Rexford, Alec Story, and David Walker. Frenetic: A Network Programming Language. *ACM SIGPLAN Notices*, 46(9), Sep. 2011.

[54] Cloud Native Computing Foundation. Istio, 2021-09-09. `https://istio.io`.

[55] Open Networking Foundation. Intent framework. ONOS, 2018. `https://wiki.onosproject.org/display/ONOS/Intent+Framework`.

[56] Open Networking Foundation. ONOS (Open Network Operating System), 2018. `https://onosproject.org/`.

[57] The OpenDaylight Foundation. Opendaylight. `https://www.opendaylight.org/`, 2013.

[58] Wireshark Foundation. Wireshark. `https://www.wireshark.org/`, 1998.

[59] Dewang Gedia and Levi Perigo. Performance evaluation of sdn-vnf in virtual machine and container. In *IEEE Conference on Network Function Virtualization and Software Defined Networks*, NFV-SDN, 2018.

[60] Wei Gong and Jiangchuan Liu. Roarray: Towards more robust indoor localization using sparse recovery with commodity wifi. *IEEE Transactions on Mobile Computing*, 18(6), 2019.

[61] M. G. Gouda and X. . A. Liu. Firewall design: consistency, completeness, and compactness. In *Proceedings of the International Conference on Distributed Computing Systems*, ICDCS, 2004.

[62] Sanket Goutam, William Enck, and Bradley Reaves. Hestia: Simple least privilege network policies for smart homes. In *Proceedings of the 12th Conference on Security and Privacy in Wireless and Mobile Networks*, WiSec '19, pages 215–220, 2019.

[63] Stephen Gutz, Alec Story, Cole Schlesinger, and Nate Foster. Splendid Isolation: A Slice Abstraction for Software-Defined Networks. In *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, HotSDN, 2012.

[64] Wonkyu Han, Ziming Zhao, Adam Doupé, and Gail-Joon Ahn. Honeymix: Toward sdn-based intelligent honeynet. In *Proceedings of the 2016 ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization*, pages 1–6. ACM, 2016.

[65] Timothy L. Hinrichs, N. Gude, M. Casado, John C. Mitchell, and S. Shenker. Practical declarative network management. In *Proceedings of the ACM Workshop on Research on Enterprise Networking*, WREN, 2009.

[66] Graham Holmes. Evolution of attacks on cisco ios devices. `https://blogs.cisco.com/security/evolution-of-attacks-on-cisco-ios-devices`, 2015.

[67] Sungmin Hong, R. Baykov, Lei Xu, Srinath Nadimpalli, and G. Gu. Towards SDN-Defined Programmable BYOD (Bring Your Own Device) Security. In *Proceedings of the Network and Distributed System Security Symposium*, NDSS, 2016.

[68] Amir Houmansadr, Chad Brubaker, and Vitaly Shmatikov. The parrot is dead: Observing unobservable network communications. In *Proceedings - 2013 IEEE Symposium on Security and Privacy, SP 2013*, pages 65–79, 8 2013.

[69] The White House. Executive order on improving the nation's cybersecurity, 2021. `https://www.whitehouse.gov/briefing-room/presidential-actions/2021/05/12/executive-order-on-improving-the-nations-cybersecurity/`.

[70] V. C. Hu, D. R. Kuhn, D. F. Ferraiolo, and J. Voas. Attribute-Based Access Control. *Computer*, 48(2), Feb 2015.

[71] Isovalent. Cilium, 2021-09-09. `https://cilium.io`.

[72] J. H. Jafarian, E. Al-Shaer, and Q. Duan. An effective address mutation approach for disrupting reconnaissance attacks. *IEEE Transactions on Information Forensics and Security*, 10:2562–2577, Dec 2015.

[73] Nitin Rao James Allworth, Alon Gavrielov. Cloudflare for offices, 2021-09-09. `https://blog.cloudflare.com/cloudflare-for-offices/`.

[74] Samuel Jero, William Koch, Richard Skowyra, Hamed Okhravi, Cristina Nita-Rotaru, and David Bigelow. Identifier Binding Attacks and Defenses in Software-Defined Networks. In *Proceedings of the USENIX Security Symposium*, 2017.

[75] Akash Shah Joshua Roberts. Policy Machine Core. GitHub, 2019. `https://github.com/PM-Master/policy-machine-core`.

[76] Jyh-Cheng Chen and Yu-Ping Wang. Extensible authentication protocol (EAP) and IEEE 802.1x: tutorial and empirical experience. *IEEE Communications Magazine*, 43(12), 2005.

[77] N. Kang, O. Rottenstreich, S. G. Rao, and J. Rexford. Alpaca: Compact Network Policies With Attribute-Encoded Addresses. *IEEE/ACM Transactions on Networking*, 25(3), June 2017.

[78] Charalampos Katsis, Fabrizio Cicala, Dan Thomsen, Nathan Ringo, and Elisa Bertino. Can I Reach You? Do I Need To? New Semantics in Security Policy Specification and Testing. In *Proceedings of the ACM Symposium on Access Control Models and Technologies*, SACMAT, 2021.

[79] Peyman Kazemian, George Varghese, and Nick McKeown. Header space analysis: Static checking for networks. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, NSDI'12, page 9, USA, 2012. USENIX Association.

[80] Peyman Kazemian, George Varghese, and Nick McKeown. Header Space Analysis: Static Checking for Networks. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation*, NSDI, 2012.

[81] Ahmed Khurshid, Xuan Zou, Wenxuan Zhou, Matthew Caesar, and P. Brighten Godfrey. Veriflow: Verifying network-wide invariants in real time. In *Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, pages 15–27, Lombard, IL, 2013. USENIX.

[82] Hyojoon Kim, Joshua Reich, Arpit Gupta, Muhammad Shahbaz, Nick Feamster, and Russ Clark. Kinetic: Verifiable Dynamic Network Control. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation*, NSDI, 2015.

[83] Hyojoon Kim, A. Voellmy, Sam Burnett, N. Feamster, and R. Clark. Lithium: Event-Driven Network Control. Georgia Tech Library, 2012. `https://smartech.gatech.edu/handle/1853/43377`.

[84] Tiffany Hyun-Jin Kim, Cristina Basescu, Limin Jia, Soo Bum Lee, Yih-Chun Hu, and Adrian Perrig. Lightweight source authentication and path validation. In *Proceedings of the ACM Conference on SIGCOMM*, pages 271–282, 2014.

[85] John Kindervag. Build Security Into Your Network's DNA: The Zero Trust Network Architecture. Security & Risk Professionals, 2020. `http://www.virtualstarmedia.com/downloads/Forrester_zero_trust_DNA.pdf`.

[86] Shashi Kiran. Data-Center: Micro-segmentation: Enhancing Security and Operational Simplicity with Cisco ACI. CISCO, 2015. `https://blogs.cisco.com/datacenter/microsegmentation`.

[87] Michael Kranch and Joseph Bonneau. Upgrading https in mid-air: An empirical study of strict transport security and key pinning. In *22nd Network and Distributed System Security Symposium NDSS*, 2015.

[88] Katharina Krombholz, Wilfired Mayer, Martin Schmiedecker, and Edgar Weippl. "i have no idea what i'm doing" - on the usability of deploying https. In *26th USENIX Security Symposium (USENIX Security 17)*. USENIX Association, August 2017.

[89] M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *Proc. 23rd International Conference on Computer Aided Verification (CAV'11)*, volume 6806, pages 585–591. Springer, 2011.

[90] Open Networking Lab. Pox wiki. `https://openflow.stanford.edu/display/ONL/POX+Wiki.html#POXWiki-InstallingPOX`, 2018.

[91] Dan Levin, Marco Canini, Stefan Schmid, Fabian Schaffert, and Anja Feldmann. Panopticon: Reaping the benefits of incremental SDN deployment in enterprise networks. In *2014 USENIX Annual Technical Conference (USENIX ATC 14)*, pages 333–345, Philadelphia, PA, 2014.

[92] F. Li, A. Lai, and D. Ddl. Evidence of advanced persistent threat: A case study of malware for political espionage. In *2011 6th International Conference on Malicious and Unwanted Software*, pages 102–109, Oct 2011.

[93] Q. Li, X. Zou, Q. Huang, J. Zheng, and P. P. C. Lee. Dynamic packet forwarding verification in sdn. *IEEE Transactions on Dependable and Secure Computing*, pages 1–1, 2018.

[94] Z. Ling, J. Luo, Y. Zhang, Ming Yang, X. Fu, and W. Yu. A novel network delay based side-channel attack: Modeling and defense. In *2012 Proceedings IEEE INFOCOM*, pages 2390–2398, March 2012.

[95] Richard Lippmann, Kyle Ingols, Chris Scott, Keith Piwowarski, Kendra Kratkiewicz, Mike Artz, and Robert Cunningham. Validating and Restoring Defense in Depth Using Attack Graphs. In *Proceedings of the IEEE Military Communications conference*, MILCOM, 2006.

[96] A. X. Liu. Formal verification of firewall policies. In *2008 IEEE International Conference on Communications*, 2008.

[97] C. Lorenz, D. Hock, J. Scherer, R. Durner, W. Kellerer, S. Gebert, N. Gray, T. Zinner, and P. Tran-Gia. An sdn/nfv-enabled enterprise network architecture offering fine-grained security policy enforcement. *IEEE Communications Magazine*, 55:217–223, March 2017.

[98] Meng Luo, Pierre Laperdrix, Nima Honarmand, and Nick Nikiforakis. Time does not heal all wounds: A longitudinal analysis of security-mechanism support in mobile browsers. In *26th Network and Distributed System Security Symposium (NDSS)*, 2019.

[99] Duohe Ma, Cheng Lei, Liming Wang, Hongqi Zhang, Zhen Xu, and Meng Li. A self-adaptive hopping approach of moving target defense to thwart scanning attacks. In *Information and Communications Security*, pages 39–53. Springer International Publishing, 2016.

[100] A. Mayer, A. Wool, and E. Ziskind. Fang: a firewall analysis engine. In *Proceedings of the IEEE Symposium on Security and Privacy*, S&P, 2000.

[101] 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.*, 38(2), 2008.

[102] Peter Mell, James M. Shook, and Serban Gavrila. Restricting Insider Access Through Efficient Implementation of Multi-Policy Access Control Systems. In *Proceedings of the ACM CCS International Workshop on Managing Insider Security Threats*, MIST, 2016.

[103] Microsoft 365 Defender Research Team. Analyzing Solorigate, the compromised DLL file that started a sophisticated cyberattack, and how Microsoft Defender helps protect customers. Microsoft Threat Intelligence Center (MSTIC), 2020. `https://www.microsoft.com/security/blog/2020/12/18/analyzing-solorigate-the-compromised-dll-file-that-started-a/sophisticated-cyberattack-and-how-microsoft-defender-helps-protect/`.

[104] Daniel Miessler. amass- automated attack surface mapping. `https://danielmiessler.com/study/amass/`, 2019.

[105] Stanislav Miskovic, Gene Moo Lee, Yong Liao, and Mario Baldi. Appprint: automatic fingerprinting of mobile applications in network traffic. In *International Conference on Passive and Active Network Measurement*, pages 57–69. Springer, 2015.

[106] Alper Tugay Mizrak, Yu-Chung Cheng, Keith Marzullo, and Stefan Savage. Detecting and isolating malicious routers. *IEEE Trans. Dependable Secur. Comput.*, 3:230–244, July 2006.

[107] Reham Mohamed, Terrance O'Connor, Markus Miettinen, William Enck, and Ahmad-Reza Sadeghi. Honeyscope: Iot device protection with deceptive network views,. In *Autonomous Cyber Deception: Reasoning, Adaptive Planning, and Evaluation of HoneyThings*. Springer International Publishing, 2019.

[108] Christopher Monsanto, Joshua Reich, Nate Foster, Jennifer Rexford, and David Walker. Composing Software-Defined Networks. In *USENIX Symposium on Networked Systems Design and Implementation*, NSDI, 2013.

[109] A. Moubayed, A. Refaey, and A. Shami. Software-Defined Perimeter (SDP): State of the Art Secure Solution for Modern Networks. *IEEE Network*, 33(5), Sep. 2019.

[110] Muhammad Mujib and Riri Fitri Sari. Design of implementation of a zero trust approach to network micro-segmentation. *International JOURNAL of Advanced Science and Technology*, 29(7), apr 2020.

[111] Geetha Nandikotkur. Evolution of attacks on cisco ios devices. `https://www.bankinfosecurity.com/200000-cisco-network-switches-reportedly-hacked-a-10788`, 2018.

[112] Ankur Kumar Nayak, Alex Reimers, Nick Feamster, and Russ Clark. Resonance: Dynamic access control for enterprise networks. In *Proceedings of the 1st ACM Workshop on Research on Enterprise Networking*, WREN '09, 2009.

[113] Ankur Kumar Nayak, Alex Reimers, Nick Feamster, and Russ Clark. Resonance: Dynamic Access Control for Enterprise Networks. In *Proceedings of the ACM Workshop on Research on Enterprise Networking*, WREN, 2009.

[114] Flowmon Networks. Flowmon: Driving network visibility. `https://www.flowmon.com/en/`, 2019.

[115] K. Neupane, R. Haddad, and L. Chen. Next generation firewall for network security: A survey. In *Proceedings of the SoutheastCon*, SECON, 2018.

[116] David M. Nicol and Vikas Mallapura. Modeling and analysis of stepping stone attacks. In *Proceedings of the Winter Simulation Conference*, WSC, 2014.

[117] Tj OConnor, William Enck, W. Michael Petullo, and Akash Verma. Pivotwall: Sdn-based information flow control. In *Proceedings of the Symposium on SDN Research*, SOSR '18, pages 3:1–3:14, 2018.

[118] The University of Adelaide. The internet topology zoo, 2010.

[119] Executive Office of the President. Moving the U.S. Government Toward Zero Trust Cybersecurity Principles. Memorandum, 2022. `https://www.whitehouse.gov/wp-content/uploads/2022/01/M-22-09.pdf`.

[120] Venkata N. Padmanabhan and Daniel R. Simon. Secure traceroute to detect faulty or malicious routing. *SIGCOMM Comput. Commun. Rev.*, page 77–82, January 2003.

[121] Kyungmin Park, Samuel Woo, Daesung Moon, and Hoon Choi. Secure cyber deception architecture and decoy injection to mitigate the insider threat. *Symmetry*, 10:14, 01 2018.

[122] Zafar Ayyub Qazi, Cheng-Chun Tu, Luis Chiang, Rui Miao, Vyas Sekar, and Minlan Yu. Simple-fying middlebox policy enforcement using sdn. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, pages 27–38, 2013.

[123] Scott Rose, Oliver Borchert, Stu Mitchell, and Sean Connelly. Zero trust architecture. National Institute of Standards and Technology, 2019. `https://csrc.nist.gov/publications/detail/sp/800-207/final`.

[124] Scott Rose, Oliver Borchert, Stu Mitchell, and Sean Connelly. Zero trust architecture, 2020-06. `https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-207.pdf`.

[125] Neil C. Rowe, E. John Custy, and Binh T. Duong. Defending cyberspace with fake honeypots, 2007.

[126] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *Computer*, 29(2), 1996.

[127] Ravi Sandhu. Roles versus Groups. In *Proceedings of the ACM Workshop on Role-Based Access Control*, RBAC, 1996.

[128] Cliff Saran. Vodafone upgrades global network with SDN for flexible scalability. *Computer Weekly*, July 2022.

[129] F. B. Schneider. Least privilege and more [computer security]. *IEEE Security & Privacy*, 1(5), 2003.

[130] Roei Schuster, Vitaly Shmatikov, and Eran Tromer. Beauty and the burst: Remote identification of encrypted video streams. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 1357–1374, Vancouver, BC, August 2017.

[131] Arash Shaghaghi, Mohamed Ali Kaafar, and Sanjay Jha. Wedgetail: An intrusion prevention system for the data plane of software defined networks. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, pages 849–861. ACM, 2017.

[132] Rob Sherwood, Michael Chan, Adam Covington, Glen Gibb, Mario Flajslik, Nikhil Handigol, Te-Yuan Huang, Peyman Kazemian, Masayoshi Kobayashi, Jad Naous, and et al. Carving Research Slices out of Your Production Networks with OpenFlow. *SIGCOMM Comput. Commun. Rev.*, 40(1), January 2010.

[133] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J.M. Wing. Automated generation and analysis of attack graphs. In *Proceedings of the IEEE Symposium on Security and Privacy*, S&P, 2002.

[134] Bill Snyder. Snowden: The nsa planted backdoors in cisco products. `https://www.infoworld.com/article/2608141/snowden--the-nsa-planted-backdoors-in-cisco-products.html`, 2014.

[135] Joel Sommers and Paul Barford. Self-configuring network traffic generation. In *Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement*, pages 68–81. ACM, 2004.

[136] K. Sripanidkulchai, C. Issariyapat, and K. Meesublak. Inference of network-wide VLAN usage in small enterprise networks. In *Proceedings of the IEEE INFOCOM Workshops*, 2008.

[137] J. Suh, T. T. Kwon, C. Dixon, W. Felter, and J. Carter. Opensample: A low-latency, sampling-based measurement platform for commodity sdn. In *2014 IEEE 34th International Conference on Distributed Computing Systems*, pages 228–237, June 2014.

[138] R. Talpade, G. Kim, and S. Khurana. NOMAD: traffic-based network monitoring framework for anomaly detection. In *Proceedings of the IEEE International Symposium on Computers and Communications*, 1999.

[139] Hacker Target. Simplify the security assessment process with hosted vulnerability scanners. `https://hackertarget.com/`, 2019.

[140] Mininet Team. Mininet an instant virtual network on your laptop (or other pc). `http://mininet.org/`, 2018.

[141] Kashyap Thimmaraju, Bhargava Shastry, Tobias Fiebig, Felicitas Hetzelt, Jean-Pierre Seifert, Anja Feldmann, and Stefan Schmid. Reigns to the cloud: Compromising cloud systems via the data plane. *CoRR*, 2016.

[142] Kashyap Thimmaraju, Bhargava Shastry, Tobias Fiebig, Felicitas Hetzelt, Jean-Pierre Seifert, Anja Feldmann, and Stefan Schmid. Reigns to the cloud: Compromising cloud systems via the data plane. *CoRR*, abs/1610.08717, 2016.

[143] Dan Thomsen and Elisa Bertino. Network Policy Enforcement Using Transactions: The NEUTRON Approach. In *Proceedings of the ACM on Symposium on Access Control Models and Technologies*, SACMAT, 2018.

[144] N. L. M. van Adrichem, C. Doerr, and F. A. Kuipers. Opennetmon: Network monitoring in openflow software-defined networks. In *2014 IEEE Network Operations and Management Symposium (NOMS)*, pages 1–8, May 2014.

[145] Romans Vanickis, Paul Jacob, Sohelia Dehghanzadeh, and Brian Lee. Access control policy enforcement for Zero-Trust-Networking. In *Proceedings of the Irish Signals and Systems Conference*, ISSC, 2018.

[146] K. V. Vishwanath and A. Vahdat. Swing: Realistic and responsive network traffic generation. *IEEE/ACM Transactions on Networking*, 17:712–725, June 2009.

[147] H. Wang, L. Xu, and G. Gu. Floodguard: A dos attack prevention extension in software-defined networks. In *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 239–250, June 2015.

[148] Rory Ward and Betsy Beyer. BeyondCorp: A New Approach to Enterprise Security. *;login:*, 39(6), 2014.

[149] Barton Whaley. Toward a general theory of deception. *The Journal of Strategic Studies*, 5(1):178–192, 1982.

[150] WonderNetwork. Global ping statistics, 2022. https://wondernetwork.com/pings.

[151] A. Wool. A quantitative study of firewall configuration errors. *Computer*, 37(6), 2004.

[152] GenShen Ye. 75,000+ MikroTik Routers Are Forwarding Owners' Traffic to the Attackers, How is Yours? Netlab 360, September 2018. https://blog.netlab.360.com/7500-mikrotik-routers-are-forwarding/owners-traffic-to-the-attackers-how-is-yours-en/.

[153] S. H. Yeganeh, A. Tootoonchian, and Y. Ganjali. On scalability of software-defined networking. *IEEE Communications Magazine*, 51(2), February 2013.

[154] Changhoon Yoon, Taejune Park, Seungsoo Lee, Heedo Kang, Seungwon Shin, and Zonghua Zhang. Enabling security functions with sdn: A feasibility study. *Computer Networks*, 85:19 – 35, 2015.

[155] Tianlong Yu, Seyed Fayaz, Michael Collins, Vyas Sekar, and Srinivasan Seshan. PSI: Precise Security Instrumentation for Enterprise Networks. In *Proceedings of the Network and Distributed System Security Symposium*, NDSS, 2017.

[156] Jim Yuill, Dorothy Denning, and Fred Feer. Using deception to hide things from hackers: Processes, principles, and techniques. Journal of Information Warfare, 2006.

[157] Jim Yuill, Mike Zappe, Dorothy Denning, and Fred Feer. Honeyfiles: deceptive files for intrusion detection. In *Proceedings from the Fifth Annual IEEE SMC Information Assurance Workshop, 2004.*, pages 116–122. IEEE, 2004.

[158] Bastion Zero. BastionZero's multi root zero-trust access protocol (MrZAP), 2021. https://github.com/bastionzero/whitepapers/blob/5ac531a3a3831a7995bb4319281d5da9e4bc7099/mrzap/README.md.

[159] Menghao Zhang, Guanyu Li, Lei Xu, Jun Bi, Guofei Gu, and Jiasong Bai. Control plane reflection attacks in SDNs: New attacks and countermeasures. In Michael Bailey, Thorsten Holz, Manolis Stamatogiannakis, and Sotiris Ioannidis, editors, *Research in Attacks, Intrusions, and Defenses*, pages 161–183. Springer International Publishing, 2018.

[160] H. Zhou, C. Wu, C. Yang, P. Wang, Q. Yang, Z. Lu, and Q. Cheng. Sdn-rdcd: A real-time and reliable method for detecting compromised sdn devices. *IEEE/ACM Transactions on Networking*, 26(5):2048–2061, 2018.