Role-Based Deception in Enterprise Networks

Iffat Anjum
ianjum@ncsu.edu
North Carolina State University
Raleigh, NC, USA

Mu Zhu
mzhu5@ncsu.edu
North Carolina State University
Raleigh, NC, USA

Isaac Polinsky
ipolins@ncsu.edu
North Carolina State University
Raleigh, NC, USA

William Enck
whenck@ncsu.edu
North Carolina State University
Raleigh, NC, USA

Michael K. Reiter
michael.reiter@duke.edu
Duke University
Durham, NC, USA

Munindar P. Singh
mpsingh@ncsu.edu
North Carolina State University
Raleigh, NC, USA

ABSTRACT

Historically, enterprise network reconnaissance is an active process, often involving port scanning. However, as routers and switches become more complex, they also become more susceptible to compromise. From this vantage point, an attacker can passively identify high-value hosts such as the workstations of IT administrators, C-suite executives, and finance personnel. The goal of this paper is to develop a technique to deceive and dissuade such adversaries. We propose HoneyRoles, which uses *honey connections* to 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 network flows. We design a prototype implementation of HoneyRoles using an OpenFlow SDN controller and evaluate its security 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.

CCS CONCEPTS

• **Security and privacy** → **Network security**; • **Networks** → *Network reliability*; *Programmable networks.*

ACM Reference Format:
Iffat Anjum, Mu Zhu, Isaac Polinsky, William Enck, Michael K. Reiter, and Munindar P. Singh. 2021. Role-Based Deception in Enterprise Networks. In *Proceedings of the Eleventh ACM Conference on Data and Application Security and Privacy (CODASPY '21), April 26–28, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3422337.3447824

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) [10]. 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 [14, 60]. These targets include emerging Software Defined Networking (SDN) switches, which provide much broader and more flexible functionality [5, 8, 50, 55]. Prior solutions [40, 45] seeking to defend against malicious forwarding devices are not directly applicable for SDN devices [63]. Furthermore, SDN data plane defenses mostly concentrate on forwarding verification and other active attacks (e.g., packet delaying, tampering, dropping) [13, 17, 25, 33, 49].

The goal of this paper 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 paper, 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

CODASPY '21, April 26–28, 2021, Virtual Event, USA

ACM ISBN 978-1-4503-8143-7/21/04…$15.00
https://doi.org/10.1145/3422337.3447824

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 [29]). 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 paper 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.

The remainder of this paper proceeds as follows. Section 2 motivates our work. Section 3 overviews HoneyRoles's architecture and major goals. Section 4 describes the design principles. Section 5 provides a security analysis using a probabilistic model checker. Section 6 evaluates performance overhead. Section 7 discusses limitations. Section 8 overviews related work. Section 9 concludes.

2 PROBLEM

Targeted attacks [18, 32] and threats to enterprise network infrastructure [22, 56] 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 [27], 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. [36] found that popular mobile web browsers failed to fully support HSTS and were left open to clickjacking attacks. Additionally, Krombholz et al. [28] 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 [13, 17, 49].

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 4.

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 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 [39, 53] 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 [15, 46].

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.

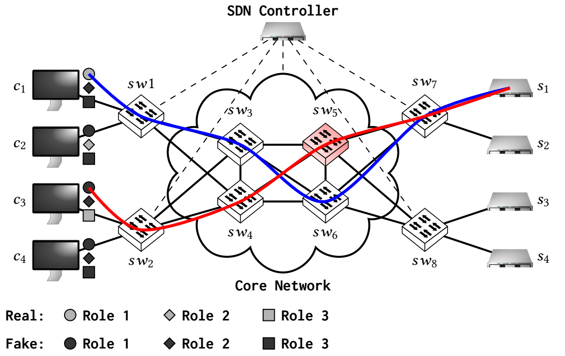


Figure 1: Overview of HoneyRoles

Figure 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 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 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 [9, 51, 58]. 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.

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.

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.

4.1.1 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 (both real and honey) via a 5-tuple: $< \texttt{ip}, \texttt{mac}, \texttt{type}, \texttt{role}, \texttt{switch} >$, 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. 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 4.1.2.

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.

4.1.2 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 6 uses Mininet [54], we emulate the existence of a honey agent by creating extra Mininet hosts (with individual IP addresses) tagged as honey hosts.

4.1.3 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 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 [19, 20, 42] use multiple network sensors distributed throughout a network. We achieve a similar capability using OpenFlow's flow-level statistics collection mechanism [52, 57]. 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 4.1.1) 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 [51], 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.

4.1.4 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 4.3) must take care when using alerts of this type.

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 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 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 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.

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 4.1.4, 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

Algorithm 1 Belief Maintenance System

```
procedure BeliefMaintenance($t$)
    #Risk update using Honey Notification
    Initialize $a_{k,t}$ & $c_{k,t}$ to 0, for all switch $s_k$
    for each entry $e \in$ Honey Notification at time $t$ do
        $\mathcal{P}_{s,d} \leftarrow getForwardingPath(e)$
        for all $k \in P_{s,d}$ do
            Increment $c_{k,t}$
            if any ATTACK logged in $e$ then
                Increment $a_{k,t}$
    for all connected switch $s_k$ do
        Update $r_{k,t}$ & $R_{k,t}$
```

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 $R_{k,0} = r_{k,0}$):

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

For convergence, $0 < \beta < 1$. To reduce the weight assigned to the current time interval, for our experiments in Section 5, we use $\beta = 0.2$; however, we have experimented with other values of β (≤ 0.5) and anecdotally found similar results. Algorithm 1 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.

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 knowledgeable adversary. 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.

5.1 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 [30]. It can provide an approximate value of a certain parameter by calculating all possible system paths. We use DTMC of PRISM [29], as it is more realistic for our model to consider time as discrete steps for maintaining the belief state of each switch.

Table 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, $\lvert L_{src,dst} \rvert$	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 PRISM model is constructed as the parallel composition of its modules. The behavior of each module is described by a collection of guarded commands, [ ] $guard \rightarrow p_1 : u_1 + \ldots + p_n : u_n$;. Here, the guard *guard* is a predicate over model variables. Each update action u_i describes 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.

For modeling complex network behavior using PRISM, we developed a *code generator* that takes in a system configuration and outputs a PRISM model with the 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 1. We define three PRISM modules: *Defender*, *System*, and *Adversary*.

5.1.1 Defender Module. The defender module specifies the current system state by defining a connection configuration as follows, $C \rightarrow \langle type, role, source, destination, path \rangle$. By selecting a new connection configuration, a new transition path is initiated. 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.

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.

5.1.2 *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.

Belief Update: After the adversary module takes actions (Section 5.1.3), 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 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 1. Here, if the current connection type is *honey* and $attack$ is true, the adversarial incident count (a_k) of each switch k on the current *path* is incremented.

5.1.3 *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. 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 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 belief parameter A_{belief}, which specifies a threshold of sufficient belief in observed information.

The adversary module has two phases: (1) attack, and (2) build confidence. For the attack 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_{belief}) 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.

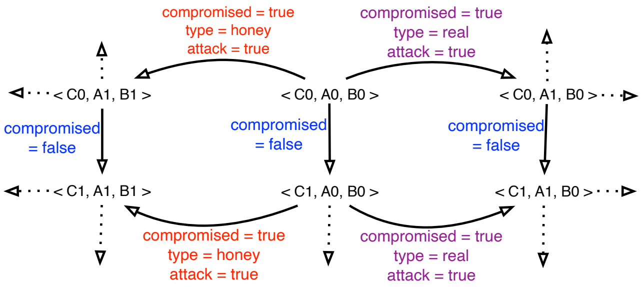


Figure 2: A simplified version of HoneyRoles Markov chain

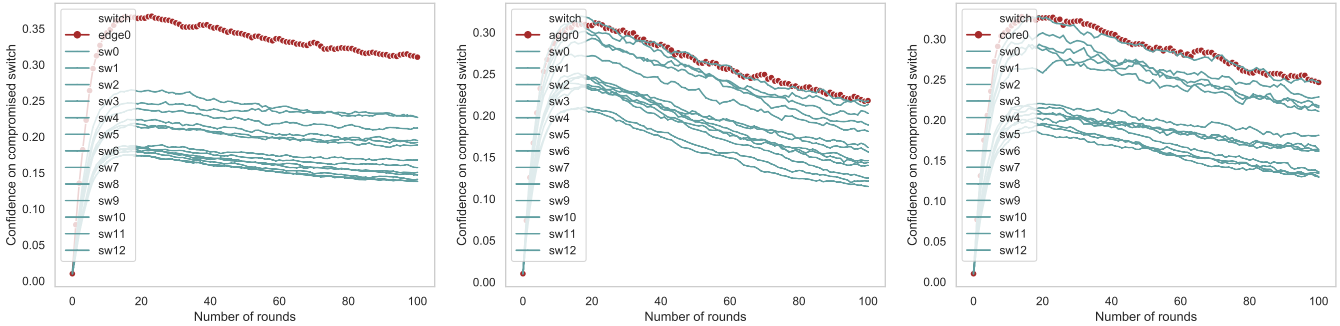
On the completion of each round, the adversary probabilistically updates A_{belief}, either increasing or decreasing it. To indicate that the adversary's knowledge is increasing with each connection, our implementation uses a higher weight (e.g., $\frac{2}{3}$) for increasing the A_{belief}. To simulate the effect of deception, we also include the possibility of decreasing the belief (e.g., $\frac{1}{3}$). Finally, we assume the adversary cannot have 100% confidence over its observation.

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 2 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) can repeat; however, this is infrequent and not shown in the figure.

5.2 Security Evaluation

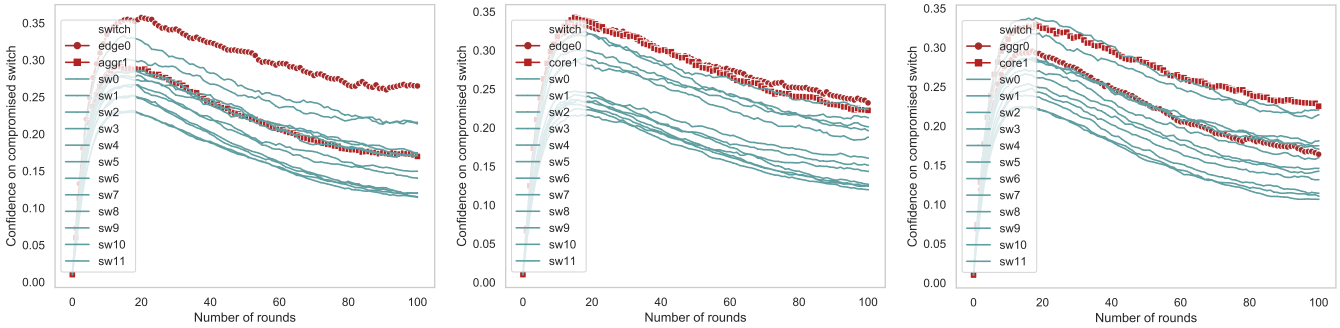
This section provides the simulation results from the PRISM module described in Section 5.1. However, first we describe our experimental setup and performance metrics.

Experimental Setup: As described in Section 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 1) and generates final PRISM logic. The "Experiment" column in Table 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 [44] and Internet2 [11] 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



(a) One compromised edge switch **(b) One compromised aggregate switch** **(c) One compromised core switch**

Figure 3: Confidence in switch compromise for one compromised switch ($\beta = 0.2$).



(a) One edge and one aggregate switch **(b) One edge and one core switch** **(c) One aggregate and one core switch**

Figure 4: Confidence in switch compromise for two compromised switches ($\beta = 0.2$).

topologies. Finally, to assess the sensitivity of β in Equation 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 [2]. 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 to generate an approximately correct result [29]. Each simulation takes 50 samples and provides the mean values as a final result. 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.

5.2.1 Detection Accuracy with One Compromised Switch. Figure 3 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 3a). This is because it is easier to isolate the attack activity over the time. As shown in Figures 3b and 3c, 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.

5.2.2 Detection Accuracy with Two Compromised Switches. Figure 4 shows three 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 5.2.1, 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 3 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

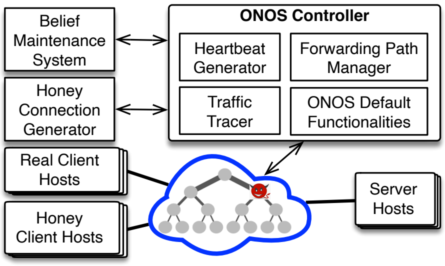


Figure 5: Experimental Layout for Evaluation

our goal. The 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.

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 [54] environment.

6.1 Implementation

Our HoneyRoles prototype is implemented as six components that comprise the design in Section 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. 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.

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 shows the network, along with the main components described in the implementation details (Section 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.

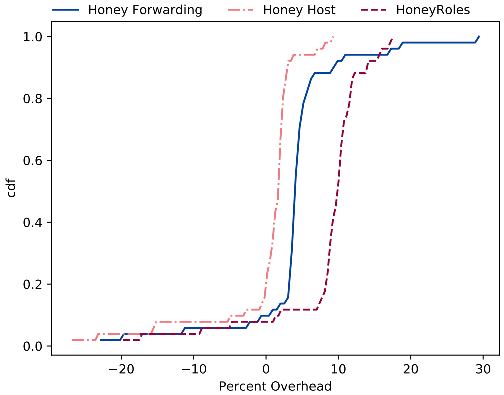


Figure 6: Percent Overhead of HTTP request completion time in each configuration compared to baseline.

To maintain consistency with our security analysis, we configured each environment with 50 real hosts. As discussed easier in Section 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.

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 6 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 6.

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 [16], 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.

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 3, we assume the existence of an ambient network traffic generator [9, 51, 58], 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 paper. 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.

8 RELATED WORK

Network reconnaissance and eavesdropping: Traditional intrusion detection systems cannot detect passive attackers performing network reconnaissance from compromised packet forwarding devices. Such reconnaissance investments are particularly apropos to advanced persistent threats (APTs) [12]. Bartlett et al. [7] 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. [48], Backes et al. [6], and Ling et al. [34] show that encrypted web traffic can leak information through packet length, packet timing, web flow size, and response delay. With increasing threats of targeted reconnaissance and attacks (e.g., Snowden [50], CISCO SYNfulKnock [22], political espionage [32]), defense against APT is becoming more critical.

Deceptive Defenses: Deception techniques provide alternative defense approach that can mislead and delay adversarial efforts, and even detect attacks in early stages. Spafford et al. [3] 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 deceptive defense solutions depend on mimicking random or static specification of system behavior, network configuration, or network infrastructures (e.g., honey-nets, decoy IP) [15, 23].

The dynamic control and programmability of an SDN environment has inspired new deception techniques. HoneyMix [21] 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 [4, 37]. For example, Achleitner et al. [1] use SDN to defend against insider reconnaissance by simulating virtual network topologies as decoys.

Software Defined Networking: SDNs have the potential to address many operational and security challenges in enterprise networks [31, 35]. 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 [38]. SDN has the potential to supplant conventional security systems [61], simplify policy enforcement [47], ensure information flow control [43], enable deceptive defense [41], and so on. 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 [55], Benton et al. [8], Menghao et al. [62]).

Network analysis and auditing tools (e.g., Header Space Analysis [24], VeriFlow [25], SDN-RDCD [63]) can protect against network or SDN controller configuration failures (or attacks). However, a compromised SDN data plane can introduce different types of attack scenarios [5], 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 [42], verifying OpenFlow events in the controller [59, 62], applying heavy-weight cryptographic approaches [26], and naive controller generated probes [13].

Sphinx [17] uses SDN control messages for incremental validation of network updates and detect suspicious behaviors (e.g., DoS, blackholing, fake topology). WedgeTail [49] 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 [33] 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 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. HoneyRoles complements the detection capabilities of prior works by adding a layer of deception to lower the effectiveness of reconnaissance in the network.

9 CONCLUSION

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 paper, 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.

ACKNOWLEDGMENTS

This research was partially sponsored by the Army Research Office and was accomplished under Grant Number W911NF-17-1-0370. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Office or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notices herein.

REFERENCES

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

[2] Gul Agha and Karl Palmskog. 2018. A Survey of Statistical Model Checking. *ACM Trans. Model. Comput. Simul.* 28, 1 (Jan. 2018), 39.

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

[4] Iffat Anjum, Mohammad Sujan Miah, Mu Zhu, Nazia Sharmin, Christopher Kiekintveld, William Enck, and Munindar P Singh. 2020. Optimizing Vulnerability-Driven Honey Traffic Using Game Theory. arXiv:cs.CR/2002.09069

[5] Markku Antikainen, Tuomas Aura, and Mikko Särelä. 2014. Spook in Your Network: Attacking an SDN with a Compromised OpenFlow Switch. In *Secure IT Systems*, Karin Bernsmed and Simone Fischer-Hübner (Eds.). Springer International Publishing.

[6] Michael Backes, Goran Doychev, and Boris Köpf. 2013. Preventing Side-Channel Leaks in Web Traffic: A Formal Approach. In *20^{th} ISOC Network and Distributed System Security Symposium.*

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

[8] Kevin Benton, L. Jean Camp, and Chris Small. 2013. OpenFlow Vulnerability Assessment. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN '13)*. ACM, 151–152.

[9] Brian M. Bowen, Vasileios P. Kemerlis, Pratap Prabhu, Angelos D. Keromytis, and Salvatore J. Stolfo. 2012. A System for Generating and Injecting Indistinguishable Network Decoys. *J. Comput. Secur.* 20, 2-3 (2012), 199–221.

[10] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. Gude, N. McKeown, and S. Shenker. 2009. Rethinking Enterprise Network Control. *IEEE/ACM Transactions on Networking* 17 (Aug 2009), 1270–1283.

[11] Internet2 Network Operations Center. 1996. Internet2. https://www.internet2.edu/

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

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

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

[15] Andrew Clark, Kun Sun, and Radha Poovendran. 2013. Effectiveness of IP address randomization in decoy-based moving target defense. In *52^{nd} IEEE Conference on Decision and Control*. 678–685.

[16] Jacob Cohen. 1988. *Statistical Power Analysis for the Behavioral Sciences* (second ed.). Routledge Member of the Taylor and Francis Group.

[17] Mohan Dhawan, Rishabh Poddar, Kshiteej Mahajan, and Vijay Mann. 2015. SPHINX: Detecting Security Attacks in Software-Defined Networks. In *ISOC Network and Distributed System Security Symposium*.

[18] R. J. Enbody and A. K. Sood. 2013. Targeted Cyberattacks: A Superset of Advanced Persistent Threats. *IEEE Security & Privacy* 11 (2013), 54–61.

[19] Rodrigo Fonseca, George Porter, Randy H. Katz, Scott Shenker, and Ion Stoica. 2007. X-trace: A Pervasive Network Tracing Framework. In *Proceedings of the 4th USENIX Conference on Networked Systems Design &#38; Implementation (NSDI'07)*. USENIX Association, 20–20.

[20] Wireshark Foundation. 1998. Wireshark. https://www.wireshark.org/.

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

[22] Graham Holmes. 2015. Evolution of attacks on Cisco IOS devices. https://blogs.cisco.com/security/evolution-of-attacks-on-cisco-ios-devices.

[23] J. H. Jafarian, E. Al-Shaer, and Q. Duan. 2015. An Effective Address Mutation Approach for Disrupting Reconnaissance Attacks. *IEEE Transactions on Information Forensics and Security* 10 (Dec 2015), 2562–2577.

[24] Peyman Kazemian, George Varghese, and Nick McKeown. 2012. Header Space Analysis: Static Checking for Networks. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation (NSDI'12)*. USENIX Association, USA, 9.

[25] Ahmed Khurshid, Xuan Zou, Wenxuan Zhou, Matthew Caesar, and P. Brighten Godfrey. 2013. 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)*. USENIX, Lombard, IL, 15–27.

[26] Tiffany Hyun-Jin Kim, Cristina Basescu, Limin Jia, Soo Bum Lee, Yih-Chun Hu, and Adrian Perrig. 2014. Lightweight Source Authentication and Path Validation. In *Proceedings of the ACM Conference on SIGCOMM*. 271–282.

[27] Michael Kranch and Joseph Bonneau. 2015. Upgrading HTTPS in Mid-Air: An Empirical Study of Strict Transport Security and Key Pinning. In *22nd Network and Distributed System Security Symposium NDSS*.

[28] Katharina Krombholz, Wilfried Mayer, Martin Schmiedecker, and Edgar Weippl. 2017. "I have No Idea What I'm Doing" - On the Usability of Deploying HTTPS. In *26th USENIX Security Symposium (USENIX Security 17)*. USENIX Association.

[29] M. Kwiatkowska, G. Norman, and D. Parker. 2011. PRISM 4.0: Verification of Probabilistic Real-time Systems. In *Proc. 23rd International Conference on Computer Aided Verification (CAV'11)*, Vol. 6806. Springer, 585–591.

[30] Marta Kwiatkowska, Gethin Norman, and David Parker. 2018. *Probabilistic Model Checking: Advances and Applications*. Springer International Publishing, 73–121.

[31] Dan Levin, Marco Canini, Stefan Schmid, Fabian Schaffert, and Anja Feldmann. 2014. Panopticon: Reaping the Benefits of Incremental SDN Deployment in Enterprise Networks. In *2014 USENIX Annual Technical Conference (USENIX ATC 14)*. Philadelphia, PA, 333–345.

[32] F. Li, A. Lai, and D. Ddl. 2011. Evidence of Advanced Persistent Threat: A case study of malware for political espionage. In *2011 6th International Conference on Malicious and Unwanted Software*. 102–109.

[33] Q. Li, X. Zou, Q. Huang, J. Zheng, and P. P. C. Lee. 2018. Dynamic Packet Forwarding Verification in SDN. *IEEE Transactions on Dependable and Secure Computing* (2018), 1–1.

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

[35] C. Lorenz, D. Hock, J. Scherer, R. Durner, W. Kellerer, S. Gebert, N. Gray, T. Zinner, and P. Tran-Gia. 2017. An SDN/NFV-Enabled Enterprise Network Architecture Offering Fine-Grained Security Policy Enforcement. *IEEE Communications Magazine* 55 (March 2017), 217–223.

[36] Meng Luo, Pierre Laperdrix, Nima Honarmand, and Nick Nikiforakis. 2019. 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)*.

[37] Duohe Ma, Cheng Lei, Liming Wang, Hongqi Zhang, Zhen Xu, and Meng Li. 2016. A Self-adaptive Hopping Approach of Moving Target Defense to thwart Scanning Attacks. In *Information and Communications Security*. Springer International Publishing, 39–53.

[38] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. 2008. OpenFlow: Enabling Innovation in Campus Networks. *SIGCOMM Comput. Commun. Rev.* 38, 2 (2008), 6.

[39] Daniel Miessler. 2019. amass- Automated Attack Surface Mapping. https://danielmiessler.com/study/amass/.

[40] Alper Tugay Mizrak, Yu-Chung Cheng, Keith Marzullo, and Stefan Savage. 2006. Detecting and Isolating Malicious Routers. *IEEE Trans. Dependable Secur. Comput.* 3 (July 2006), 230–244.

[41] Reham Mohamed, Terrance O'Connor, Markus Miettinen, William Enck, and Ahmad-Reza Sadeghi. 2019. HONEYSCOPE: IoT Device Protection with Deceptive Network Views,. In *Autonomous Cyber Deception: Reasoning, Adaptive Planning, and Evaluation of HoneyThings*. Springer International Publishing.

[42] Flowmon Networks. 2019. Flowmon: Driving Network Visibility. https://www.flowmon.com/en/.

[43] Tj OConnor, William Enck, W. Michael Petullo, and Akash Verma. 2018. PivotWall: SDN-Based Information Flow Control. In *Proceedings of the Symposium on SDN Research (SOSR '18)*. Article 3, 14 pages.

[44] The University of Adelaide. 2010. The Internet Topology Zoo. http://www.topology-zoo.org/contact.html

[45] Venkata N. Padmanabhan and Daniel R. Simon. 2003. Secure Traceroute to Detect Faulty or Malicious Routing. *SIGCOMM Comput. Commun. Rev.* (Jan. 2003), 77–82.

[46] Kyungmin Park, Samuel Woo, Daesung Moon, and Hoon Choi. 2018. Secure Cyber Deception Architecture and Decoy Injection to Mitigate the Insider Threat. *Symmetry* 10 (01 2018), 14.

[47] Zafar Ayyub Qazi, Cheng-Chun Tu, Luis Chiang, Rui Miao, Vyas Sekar, and Minlan Yu. 2013. SIMPLE-fying Middlebox Policy Enforcement Using SDN. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*. 27–38.

[48] Roei Schuster, Vitaly Shmatikov, and Eran Tromer. 2017. Beauty and the Burst: Remote Identification of Encrypted Video Streams. In *26th USENIX Security Symposium (USENIX Security 17)*. Vancouver, BC, 1357–1374.

[49] Arash Shaghaghi, Mohamed Ali Kaafar, and Sanjay Jha. 2017. 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*. ACM, 849–861.

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

[51] Joel Sommers and Paul Barford. 2004. Self-configuring Network Traffic Generation. In *Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement*. ACM, 68–81.

[52] J. Suh, T. T. Kwon, C. Dixon, W. Felter, and J. Carter. 2014. OpenSample: A Low-Latency, Sampling-Based Measurement Platform for Commodity SDN. In *2014 IEEE 34th International Conference on Distributed Computing Systems*. 228–237.

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

[54] Mininet Team. 2018. Mininet An Instant Virtual Network on your Laptop (or other PC). http://mininet.org/.

[55] Kashyap Thimmaraju, Bhargava Shastry, Tobias Fiebig, Felicitas Hetzelt, Jean-Pierre Seifert, Anja Feldmann, and Stefan Schmid. 2016. Reigns to the Cloud: Compromising Cloud Systems via the Data Plane. *CoRR* (2016). arXiv:1610.08717

[56] Kashyap Thimmaraju, Bhargava Shastry, Tobias Fiebig, Felicitas Hetzelt, Jean-Pierre Seifert, Anja Feldmann, and Stefan Schmid. 2016. Reigns to the Cloud: Compromising Cloud Systems via the Data Plane. *CoRR* abs/1610.08717 (2016).

[57] N. L. M. van Adrichem, C. Doerr, and F. A. Kuipers. 2014. OpenNetMon: Network monitoring in OpenFlow Software-Defined Networks. In *2014 IEEE Network Operations and Management Symposium (NOMS)*. 1–8.

[58] K. V. Vishwanath and A. Vahdat. 2009. Swing: Realistic and Responsive Network Traffic Generation. *IEEE/ACM Transactions on Networking* 17 (June 2009), 712–725.

[59] H. Wang, L. Xu, and G. Gu. 2015. FloodGuard: A DoS Attack Prevention Extension in Software-Defined Networks. In *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. 239–250.

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

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

[62] Menghao Zhang, Guanyu Li, Lei Xu, Jun Bi, Guofei Gu, and Jiasong Bai. 2018. Control Plane Reflection Attacks in SDNs: New Attacks and Countermeasures. In *Research in Attacks, Intrusions, and Defenses*, Michael Bailey, Thorsten Holz, Manolis Stamatogiannakis, and Sotiris Ioannidis (Eds.). Springer International Publishing, 161–183.

[63] H. Zhou, C. Wu, C. Yang, P. Wang, Q. Yang, Z. Lu, and Q. Cheng. 2018. SDN-RDCD: A Real-Time and Reliable Method for Detecting Compromised SDN Devices. *IEEE/ACM Transactions on Networking* 26, 5 (2018), 2048–2061.