Identifier Binding Attacks and Defenses in Software-Defined Networks *

Samuel Jero
Purdue University
sjero@purdue.edu

William Koch
Boston University
wfkoch@bu.edu

Richard Skowyra
MIT Lincoln Laboratory
richard.skowyra@ll.mit.edu

Hamed Okhravi
MIT Lincoln Laboratory
hamed.okhravi@ll.mit.edu

Cristina Nita-Rotaru
Northeastern University
c.nitarotaru@neu.edu

David Bigelow
MIT Lincoln Laboratory
dbigelow@ll.mit.edu

Abstract

In this work, we demonstrate a novel attack in SDN networks, Persona Hijacking, that breaks the bindings of all layers of the networking stack and fools the network infrastructure into believing that the attacker is the legitimate owner of the victim's identifiers, which significantly increases persistence. We then present a defense, SECUREBINDER, that prevents identifier binding attacks at all layers of the network by leveraging SDN's data and control plane separation, global network view, and programmatic control of the network, while building upon IEEE 802.1x as a root of trust. To evaluate its effectiveness we both implement it in a testbed and use model checking to verify the guarantees it provides.

1 Introduction

Modern networks use various identifiers to specify entities at network stack layers. These identifiers include addresses, like IP addresses or MAC addresses, and domain names, as well as less explicitly known values such as the switch identifier and the physical switch port to which a machine is connected. Identifiers are used in modern networks not only to establish traffic flow and deliver packets, but also to enforce security policies such as in firewalls or network access control systems [3]. In order to achieve proper operation and security guarantees, network infrastructure devices (*e.g.,* switches, network servers, and SDN controllers) implicitly or explicitly associate various identifiers of the same entity with each other in a process we call *identifier binding*. For example, when a host acquires an IP address from a DHCP server, the server *binds* that IP address to the host's MAC address; when an ARP reply is sent in response to an ARP request, the source host binds the IP address in the ARP request to the MAC address in the ARP reply.

Given the importance of these identifier bindings, it is not surprising that numerous attacks against them have been developed, including DNS spoofing [48], ARP poisoning [29], DHCP forgery, and host location hijacking [24]. These attacks are facilitated by several network design characteristics: (1) reliance on insecure protocols that use techniques such as broadcast for requests and responses without any authentication mechanisms, (2) allowing binding changes without considering the network-wide impact of services relying on them, (3) allowing independent bindings across different layers without any attempt to check consistency, and (4) allowing high-level changes to identifiers that are designed and assumed to be unique. Numerous defenses have also been proposed to prevent identifier binding attacks of various types in traditional networks [18, 48, 2].

Software-Defined Networking (SDN) is a new networking paradigm that facilitates network management and administration by providing an interface to control network infrastructure devices (*e.g.,* switches). In this paradigm, the system responsible for making traffic path decisions (the control plane) is separated from the switches responsible for delivering the traffic to the destination (the data plane). The SDN controller is the centralized system that manages the switches, installs forwarding rules, and presents an abstract view of the network to SDN applications. SDN provides flexibility, manageability, and programmability for network administrators. Although previous work has focused on various aspects of the intersection of security and SDNs [46, 27, 28, 45, 26, 36, 44, 5, 14, 52], there has been little work on studying identifier binding attacks and their implications in SDN systems.

In this paper, we first study identifier binding attacks in SDN systems. We show that the centralized control exacerbates the implications and consequences of weak identifier binding. This allows malicious hosts to poi-

*This work is sponsored by the Department of Defense under Air Force Contract #FA8721-05-C-0002. Opinions, interpretations, conclusions and recommendations are those of the author and are not necessarily endorsed by the United States Government.

son identifier bindings not only in their own broadcast domain, as is the case with many identifier binding attacks in traditional networks, but also in the entire SDN network. Moreover, we show that, unlike traditional networks where identifier binding attacks are limited to a small subset of identifiers, in SDN, identifier binding attacks can be so severe that they allow complete takeover of *all* network identifiers of the victim host at once, in an attack we dub *Persona Hijacking*. More damagingly, in a Persona Hijacking attack the malicious host fools the network infrastructure devices into believing that it is the legitimate owner of the victim's identifiers, allowing it to persistently hold the compromised identifiers. Our attack succeeds even in the presence of the latest secure SDN solutions such as TopoGuard [24], SPHINX [14], and SE-Floodlight [45].

We then show how the SDN design philosophies of programmable infrastructure, separation of control and data planes, and centralized control can be used to prevent identifier binding attacks. We design and implement a defense, SECUREBINDER, to establish strong bindings between various network identifiers. First, we extend the 802.1x protocol to establish a root-of-trust for strong authentication of a machine. Building on this root-of-trust, we then implement additional components of the defense to strongly bind higher-level identifiers to the MAC address. As part of SECUREBINDER, we force all identifier binding broadcast traffic to go through the control plane and program switches to drop all data plane broadcasts, preventing the hijacking of higher-level identifiers (IP or domain name) bound to a given MAC address. Our solution does not require any changes on the end-hosts.

We extensively evaluate the effectiveness of our defense experimentally, using testbed implementations of various identifier binding attacks, as well as formally, using model checking. Our experimental and formal evaluations indicate that SECUREBINDER can successfully stop identifier binding attacks while incurring a small overhead, mainly in the form of additional network join latency due to the initial authentication step.

Roadmap. In section 2, we discuss the key bindings in a modern network stack. In section 3 we describe the Persona Hijacking attack. We present our defense in section 4, then evaluate it formally and experimentally in section 5. We discuss limitations of our attack and defense in section 6 and then consider related work in section 7 before concluding in section 8.

2 Identifier Binding

In this section we provide an overview of the main identifiers used at different network layers and discuss identifier binding attacks in traditional networks and what makes these attacks more dangerous in SDN networks.

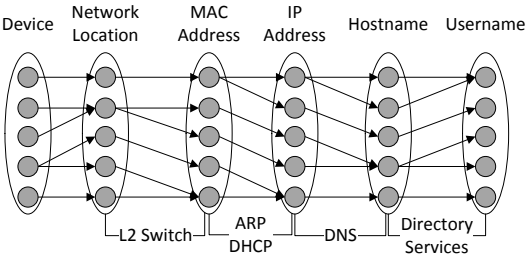

Figure 1: Network Identifier Bindings. Protocols mediating the binding for IPv4 are shown at bottom.

2.1 Overview of Identifier Bindings

Network protocols rely on identifier binding in order to operate correctly and efficiently. Because of the stack model where layers can access services only from adjacent layers, identifier binding takes place in two forms, *explicit* – achieved through network protocols or manual configuration, and *implicit* – achieved through already existing mappings. Below we describe the identifier bindings that are critical to correct functioning of a network (see Fig. 1). For additional background on the identifiers discussed here, please see Appendix A.

Network location to device: In traditional networks, a device's network location is represented implicitly by the switch and port that packet forwarding rules are bound to and the location of ACL rules or other configuration specific to that device.

MAC address to network location: Binding MAC addresses to network locations is done implicitly based on observed network traffic by Layer-2 switches; the source MAC address of traffic is used to learn which network port on a switch corresponds to that MAC address.

IP address to MAC address: Mapping a unicast IP address to a MAC address is usually done via ARP for IPv4 and NDP for IPv6. These mechanisms broadcast a query asking who has a given IP address and the device with that IP address unicasts a response including its MAC address. An interface with a single MAC address may have more than one IP address associated with it.

Host names to IP addresses and IP addresses to hostnames: Hostnames are mapped to IP addresses in several ways. The most common is unicast, centrally configured DNS. However, multicast DNS (mDNS) without a central server also exists, as does a legacy naming service for Microsoft Windows known as NETBIOS. Note that DNS and mDNS can also be used for reverse resolution: finding a hostname given an IP address. A single IP address may be associated with multiple hostnames and a single hostname may be associated with multiple IP addresses.

In unicast DNS, the hostname to IP bindings are either manually configured by an administrator or automatically updated using the DNS update command. The protocol for automatically updating hostnames is Dynamic

DNS [51] and lacks authentication. A secure version [53] exists, but is rarely used. Microsoft Active Directory has its own scheme to authenticate DNS updates to AD-integrated DNS servers while other directory services have the DHCP server update the DNS records for clients when they acquire IP addresses [19]. To map a hostname to an IP address, unicast DNS uses UDP (although TCP can also be used) to send a request to the DNS server. Responses are returned in the same way and contain no authentication. DNSSEC [2] cryptographically authenticates DNS responses, preventing modification or forgery, but is rarely deployed. Both Multicast DNS (mDNS) and NETBIOS rely on unauthenticated broadcast requests. Hosts listen for these requests and respond if they have the queried hostname. NETBIOS can use a registration server to speed up this process.

Username to hostname: This binding occurs either as per-system user accounts or via a directory service, of which Active Directory is the most prominent.

Active Directory (and its open-source counterpart Samba) is a directory service that maintains information on users, groups, access rights, and configuration information for an organization and uses this information for centralized authentication and management. It leverages LDAP for directory access and Kerberos for authentication. It authenticates both users and machines to the network and provides configuration management to Windows clients. Unfortunately, Active Directory machine authentication does not provide authentication of lower level network identifiers like MAC or IP addresses. Authentication is based on Kerberos, but Active Directory-issued tickets are bound only to the hostname and not to the IP address by default [17].

Directory services like Active Directory do *not* know precisely who is logged in at any given point in time. Further, a connection to the network might *not* trigger authentication. For example, if a user is already logged in when they connect to the network, the connection does not trigger authentication. For per-system user accounts, the authentication and management is local to each system and not visible to the network. Higher level protocols, like NFS, may still rely on this information.

2.2 Binding Attacks in non-SDN Networks

The ultimate goal of layer-to-layer bindings is to allow a mapping across the entire stack where higher level identifiers are mapped by transitivity to lower layers and, ultimately, to device identifiers.

Definition 1 (Identifier binding attack): We define an identifier binding attack as 1) replacing or creating a binding such that the identifiers bound together are associated with different devices, or 2) utilizing identifiers associated with a known, offline device in a binding.

There are several design factors and architectural characteristics that facilitate identifier binding attacks:

(1) Reliance on insecure protocols: Many of these bindings are constructed based on broadcast requests that query the entire broadcast domain while others are formed implicitly based on spoof-able identifiers in observed traffic. Thus, an attacker can easily impact these bindings simply by sending spoofed packets or listening for broadcast queries and responding.

(2) Treating binding creation and changes as the same operation: Once a binding is created, there are services that rely on it. Changing a binding, for example, because of a host migration or IP address reassignment, has implications on all services that rely on it network-wide. Not distinguishing between creation and changes to a binding allows an attacker to reset existing bindings simply by claiming to have an identifier.

(3) Independent changes: Many bindings are treated independently from each other with no attempts to use information recorded in one binding, for example the MAC to network location binding, to validate updates being made to others, like the IP address to MAC address binding. This enables attackers to use packets that violate one binding to successfully attack a different binding.

(4) Ability to change identifiers: Identifiers that are assumed to be unique, like MAC addresses, are actually mutable and easily changed in software. Hence, attackers can readily impersonate other devices to the network.

These characteristics enable a wide variety of attacks on identifier binding protocols, including ARP spoofing, DNS spoofing, and Rogue DHCP servers. ARP spoofing is enabled by the broadcast mechanism employed by ARP to bind IP addresses to MAC addresses as well as by bindings at different layers not being used to validate each other. In a similar manner, Rogue DHCP servers are enabled by the broadcast nature of DHCP that allows any host to listen for and respond to requests. DNS spoofing is possible because bindings are treated independently, allowing a host to use spoofed packets to send DNS responses as if they came from the legitimate DNS server.

Limitations of identifier binding attacks in traditional networks: Identifier management in IPv4 Ethernets must contend with several architectural aspects of the network stack that impact the scope, consistency, and security of identifier bindings between different network identifiers.

(1) Distributed Control State: Traditional networks maintain distributed control state in both network infrastructure (*e.g.,* switches and routers) as well as dedicated identifier management servers such as DHCP and DNS. This ensures that network layer boundaries define the scope of the relevant identifier bindings. Layer-2 switches form broadcast domains over which packets are forwarded based on MAC–Port bindings maintained by

each device. Outside of these layer boundaries, Layer-2 identifiers are overwritten by routers and forwarding is based on Layer-3 IP addresses. This effectively limits the scope of Layer-2 attacks like ARP or MAC spoofing only to that broadcast domain, as all other regions of the network are only reachable via Layer-3 routing.

(2) Intelligent Routers: Modern switches and routers have defenses to mitigate attacks on identifier bindings. These include techniques such as Cisco's Dynamic ARP Inspection and DHCP Snooping systems, which maintain local databases of identifier bindings and can drop packets based on coarse-grained heuristics and manually configured trust relationships [10]. Network interfaces which are not registered as DHCP servers, can be set to drop server-side DHCP messages such as lease offers.

(3) Rapid Rule Consistency: Modern IPv4 Ethernets rely on interior gateway protocols (*e.g.,* OSPF) to build a network information base (NIB) at every router. The NIB is used to install routing rules, which are updated whenever the NIB state changes. Since this update is local to the router, there is very little delay between a NIB update and a routing rule change to make forwarding behavior consistent with the NIB. This limits the ability of an attacker to cause blackhole or redirection attacks based on stale routing rules (though attacks on the routing protocol itself remain possible).

2.3 Binding Attacks in SDN Networks

While identifier management in SDNs uses largely the same protocols as those used by IPv4 Ethernets, the architecture of an SDN imposes different challenges to maintaining the security of identifier bindings. SDNs differ from traditional networks in three key aspects that can be used to amplify the impact of existing identifier binding attacks: a unified control plane, bare-metal switches, and delayed rule consistency.

Unified Control Plane: OpenFlow networks are divided into a separate data plane (the switches) and control plane (the controller). This unifies the entire network under a single SDN controller (or communicating set of controllers) and removes most traditional divisions of a network into broadcast domains and subnets. Protocol data structures which would normally be maintained per-switch or per-router are maintained only at the controller, and messages which would normally not leave the local switch/router are instead sent to the controller. Many controllers implement Proxy ARP, for example, in which a single master ARP table is maintained for the entire network. ARP Requests are sent to the controller, which generates an ARP Reply via a `packet_out`. Attackers can use this to their advantage. Any ARP Spoofing attack can now target any victim on the entire SDN, whereas a traditional network requires the attacker and victim to share a broadcast domain.

Bare-Metal Switches: OpenFlow switches have no internal packet-processing logic beyond the flow rules installed by a controller. Thus, defenses that have traditionally been implemented by network infrastructure (such as Dynamic ARP inspection and DHCP Snooping [10]) are not present in SDNs. Additionally, no open source SDN controller currently provides any Layer-2 or Layer-3 security systems beyond user-configurable firewall rules. As a result, attacks which are easily detected in traditional networks (such as Rogue DHCP servers) go unnoticed in vanilla OpenFlow networks.

Delayed Rule Consistency: SDN controllers implement a global NIB in order to determine what flow rules to install in response to a `packet_in` event. Approaches to populating it have a common component. During `packet_in` processing, the source IP and MAC addresses are used to update the NIB with the switch and port on which an end-host is located. Destination IP and MAC addresses are looked up in the NIB to determine the switch port on which the packet should be forwarded. A flow rule is then synchronously installed in the relevant switch(es) with match fields determined by the current NIB state before the packet is sent back to the switch for forwarding. Unfortunately, when the NIB state is updated, old flow rules are not removed from switches (probably because attempting to differentially update all flow rules on NIB updates would dramatically increase flow latency). A common work-around (used by Ryu and POX) is to set a hard or soft timeout on flow rules. Soft timeouts count down from the last time the rule was triggered, while hard timeouts count down from rule installation. When a timeout is reached, the rule is deleted. This bounds the duration that inconsistent rules can persist, but does not solve the problem on shorter timescales. Attackers can take advantage of temporarily inconsistent flow rules to intercept messages meant for another host or blackhole traffic.

We have observed the lack of traditional ARP poisoning/DHCP snooping defenses and the presence of delayed rule consistency experimentally in ONOS and Ryu and confirmed both in the Floodlight and POX source code. We leave a complete exploration of additional controllers to future work.

3 Persona Hijacking

We present an attack against identifier bindings in SDNs that allows complete takeover of *all* network identifiers of the victim host at once, an attack we dub *Persona Hijacking*. We first describe the attacker capabilities required by the attack, then describe the Persona Hijacking attack in detail.

Table 1: Impact of Identifier Binding Attacks

Attack	Spoofed ID			Persistence	Area Affected	Legitimized	Defenses	
	MAC	IP	Hostname				Non-SDN	SDN
ARP Spoofing	X	X	–	Minutes	Broadcast Domain	–	X	–
Rogue DHCP	–	X	X	Days	Subnet	–	X	–
DNS Spoofing	–	X	X	Minutes	DNS Domain	X	X	X
Persona Hijacking	X	X	X	Days	Entire Network	X	–	–

3.1 Attacker Model

We consider an enterprise IPv4 Ethernet network using the Openflow SDN architecture and a standard OpenFlow controller such as Ryu, POX, NOX, Floodlight, OpenDaylight, ONOS, or Beacon. End-hosts use ARP to look up the MAC address associated with an IP and use DHCP to obtain IP addresses from a single DHCP server. Additionally, the network has an internal DNS server for managing intranet hostnames and a directory services package such as Microsoft Active Directory.

We assume the attacker has compromised one or more end-hosts on this network and is attempting to use those hosts to impersonate a target server to clients in order to subvert additional end-hosts and move laterally in the network. OpenFlow switches and the controller are outside of adversarial control and act as trusted infrastructure. Thus, the attacker cannot perform a man-in-the-middle attack without corrupting network routing state or stealing the identifiers (*e.g.,* IP address) of the victim end-host.

We assume that the attacker has not compromised any of the critical servers (such as directory services, DNS, or DHCP). This is because the attacker would have more powerful options than the attack presented here, if such servers have been compromised.

3.2 Persona Hijacking Attacks

While several attacks against identifier bindings exist, their impact is limited in traditional networks. They impact only a single binding, persist briefly, have a limited area of effect, and many defensive solutions exist. We summarize the characteristics of the most common attacks on identifier bindings in Table 1. ARP Spoofing, for example, corrupts a MAC–IP binding within a Layer 2 broadcast domain until the ARP cache entry times out, at which point the attack must be re-launched.

These limitations do not hold for SDNs, which permit powerful new attacks on identifier bindings. We introduce one such attack, Persona Hijacking, which allows complete takeover of *all* network identifiers of the victim host at once, can persist for days, affects the entire network, and has no existing defenses. Specifically, our Persona Hijacking attack allows an attacker in an SDN-based network to take over an IP address and DNS domain name from a victim end-host by progressively breaking the MAC Address to Network Location, IP Address to MAC address, and (in some network configurations) Hostname to IP Address bindings.

A key feature of our attack, which is unachievable using traditional identifier binding attacks (*e.g.,* ARP spoofing), is that it affects the network infrastructure such that the attacker becomes the *owner of record* for the IP address. That is, the DHCP server believes that the victim's IP is bound to the *attacker's* MAC address. This allows Persona Hijacking attacks to effectively co-opt the DHCP server and propagate the deception further into the network.

Persona Hijacking consists of two main phases. The first phase, which we refer to as *IP takeover*, relies on a client-side attack against DHCP to break the IP address to MAC address and hostname to IP address bindings in order to hijack the IP address and hostname of the victim by binding both of them to the attacker's MAC address. The second phase, which we refer to as *Flow Poisoning*, exploits the delayed flow rule consistency present in SDNs to break (from the perspective of the victim) the MAC address to network location binding of the DHCP server in order to (from the perspective of the DHCP server) legitimize the first phase and make the victim appear to have willingly given up its IP address. A timeline of the complete attack is shown in Figure 2.

IP takeover Details. IP takeover operates in two steps. First, the attacker breaks the binding between the victim's IP address and MAC address by forging a DHCP_RELEASE message to make the DHCP server release the victim's IP address into the pool of available addresses. This does *not* break the hostname to IP address binding, as the recommended practice in an enterprise setting is for the *client* to manage DNS A record updates [33]. The next step is to bind the released IP address to the *attacker's* MAC address. However, the DHCP specification [15] recommends that the DHCP server should offer addresses from their unused pool before offering addresses that were recently released. Hence, the attacker mounts a partial (and temporary) DHCP starvation attack. The DHCP server is flooded with DHCP_DISCOVER messages using random MAC addresses, until the target's IP address is offered. Once the DHCP server offers the victim's IP, the attacker confirms the lease, and the starvation attack is halted. The

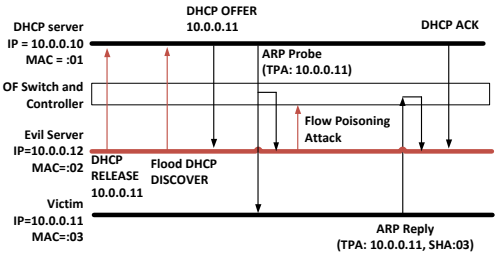

Figure 2: Timeline of the Persona Hijacking attack.

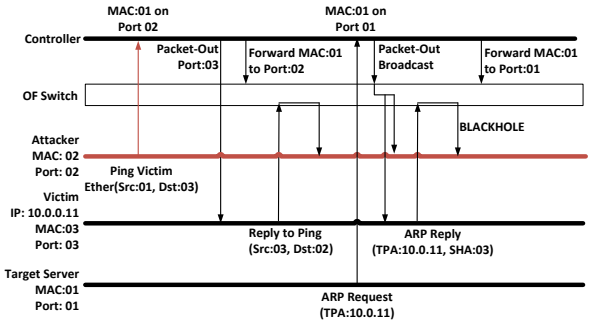

Figure 3: Timeline of the Flow Poisoning Attack

victim's IP address and hostname are now bound to the attacker's MAC address *by the DHCP server*.

Note that the DHCP_DISCOVER is the first half of the DHCP handshake. An additional confirmation from the client is required to finalize the lease, and most DHCP servers will avoid offering such addresses to other clients for a few minutes. This limits the exposure of the exhaustion attack and its impact on legitimate clients to a short time window. Furthermore in networks with high DHCP churn, an attacker can perform strategically timed and limited DHCP starvation to avoid detection by allowing new clients to consume the IP addresses in the unused pool.

IP takeover Impact: SDN controllers that use DHCP to manage forwarding rules will redirect traffic bound for the victim's IP address to the attacker's network location. This blackholes the victim, preventing them from receiving further traffic, and allows the attacker to impersonate the victim. Any client request made to the hostname associated with the stolen IP address will be sent to the attacker. Any existing connections made by the victim will continue uninterrupted. New connection requests made by the victim will reach their destination but the response packets will be sent to the attacker. Since the victim is unaware that its IP address has been re-bound, the attack lasts until the victim obtains a new DHCP lease, which would typically be hours or days later. This ensures the hijacked binding will persist over large timescales.

Flow Poisoning Details. In some cases the IP takeover phase is sufficient, and the victim's persona is successfully hijacked. However, in order to be compliant with the DHCP RFC [15], many DHCP servers probe a reused address before re-allocating it to a new client. Typically, the DHCP server sends a broadcast ARP request to see if any host claims the reused address, validating that this IP address is not in use. If this probing before re-allocating an IP address is used, the IP takeover phase will fail, because the victim is not aware that their IP address has been released by the DHCP server and will respond to the ARP request, causing the DHCP server to refuse to lease the target IP address. In order to ensure that IP takeover succeeds even in the presence of probing, we developed a Flow Poisoning attack that breaks the MAC address to network location binding of the DHCP server in order to blackhole this response.

Our Flow Poisoning attack takes advantage of a race condition unique to SDNs in order to break the MAC address to network location binding of a targeted end-host. The attack relies on the fact that SDN flow rules may be temporarily inconsistent with the NIB maintained by the controller. Since flow rules reflect the NIB state at the time of their installation and (due to scalability challenges and network latency) are not updated instantly when the NIB state changes, they may reflect previous network states that no longer hold. Two approaches are used to bound the duration of this inconsistency. The first, used by Ryu and other learning-switch-based controllers, relies on installing hard or soft timeouts on all rules. The second, used by ONOS and other controllers that implement a Host-Discovery Protocol, uses a separate monitoring thread to detect host movement and remove inconsistent rules. This monitor runs concurrently with the NIB updater responsible for generating `packet_out` events, which introduces an exploitable race condition where the message is sent prior to new flow rules being installed.

The attacker can take advantage of this temporary inconsistency to blackhole traffic from a target source s to a target destination d. Figure 3 depicts a timeline of this attack. First, the attacker sends traffic (we chose ICMP pings, but the attack is agnostic to the packet payload) to s with the source MAC address of d. This breaks the MAC address to network location binding of d.

Upon entering the switch, the forged packets cause a flow table miss (since the MAC address of d is not bound to the network location of the attacker) and are sent to the controller. The controller updates its MAC to location

binding such that the destination MAC d is bound to the attacker's switch port. It then installs a new flow rule in the switch enforcing this binding and forwards the ping to the victim s. The victim s replies to the forged ICMP ping, which causes another flow rule to be inserted into the switch that sends all traffic from s to d to the switch port on which the attacker resides.

Once d originates traffic, the controller will update its internal mappings to correctly track d's location (*i.e.,* re-binding the MAC address of d to the network location of d). This begins the race condition on which Flow Poisoning relies. The now-inconsistent flow rule which binds the attacker's location to d's MAC address will *not* be removed from the switch upon NIB update. Instead, it will either timeout several seconds later or be deleted by the separate host mobility tracking thread. Until this occurs, traffic from s to d will be sent to the attacker.

In the context of IP takeover, this technique can be used to blackhole the reachability probe conducted by a DHCP server prior to assigning the victim's IP to the attacker. Since the initial ARP request from the DHCP server is broadcast to all ports, it is not possible to blackhole. We, therefore, blackhole the unicast response from the victim to the DHCP server by breaking the DHCP server's MAC address to network location binding. Note that while the Flow Poisoning phase of the attack only lasts until the flow rules are updated, the larger Persona Hijacking attack utilizes this to create an attack that will persist until the victim's DHCP lease expires.

3.3 Attack Implementation

We have implemented both the IP takeover and Flow Poisoning phases as fully automated Python scripts running on an attacking end-host. We use Scapy[1] to forge a DHCP_RELEASE message for the IP takeover phase and to request new IP addresses until the victim's address is offered. The Flow Poisoning phase simply consists of sending an ICMP ping with the DHCP server's MAC address to the victim as soon as the DHCP Offer is received by the attacker. The entire attack was tested in an emulated SDN environment using Mininet 2.2.1 [30], against both the ONOS and Ryu controllers. An analysis of the Floodlight and POX source code suggests that they are also vulnerable.

Because Flow Poisoning relies on a race condition, we measured the Persona Hijacking success rate over 10 trials, each of which took an average of 90.39 seconds to acquire the target IP address. For Ryu, which uses hard flow rule timeouts, the success rate was 90%. Failures corresponded to an ARP Reply that was sent by the victim to the DHCP server after the inconsistent flow rule expired. For ONOS, the Flow Poisoning phase was not

[1]http://www.secdev.org/projects/scapy/

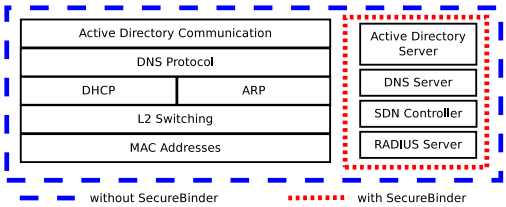

Figure 4: Components of the network that must be trusted with and without SECUREBINDER

necessary because ONOS provides a DHCP server in the controller which does not probe reused addresses before allocating them to new clients. Persona Hijacking was always successful.

Our testbed was a machine running Ubuntu 14.04 with an Intel Core i7-3740QM CPU and 16GB memory. The experimental topology consisted of three hosts directly connected to an OpenFlow 1.3 switch. One of these hosts was a DHCP server running `udhcpd` from BusyBox v1.21.1. The other two were the attacker and the victim. The DHCP server was configured to lease a total of 5 IP addresses.

4 SECUREBINDER

This section presents SECUREBINDER, our system for securing network identifier bindings. We first present our design and then describe our implementation.

4.1 Design

The Persona Hijacking attack and other identifier attacks are possible because of several network design characteristics of identifier binding in traditional networks: (1) reliance on insecure protocols using techniques like broadcast for requests and responses without any authentication mechanisms, (2) allowing the changing of bindings without considering the network-wide impact to services relying on them, (3) allowing for independent bindings across different layers without any attempt to check consistency, and (4) allowing high-level changes to identifiers that are designed and assumed to be unique. We design SECUREBINDER as a comprehensive solution to identifier binding attacks and the above attack facilitating factors, not merely as another defense against a specific attack. In doing so we dramatically reduce the number of network components that must be trusted, as shown in Figure 4.

SECUREBINDER leverages SDN and IEEE 802.1x to target the facilitating factors as follows:

• It leverages SDN functionality to separate the identifier binding control traffic from the regular data plane, isolating it from an attacker, and creating a binding mediator which can perform additional security checks on

identifier bindings. While this approach does not eliminate the use of insecure protocols for identifier binding (which would require changes to every end host), it does remove the requirement to trust these protocols.

• It validates identifier binding changes, by leveraging the global view of the network and its identifiers that SDN provides. It distinguishes between creating new bindings and changing existing ones, requiring validation that the old binding is no longer active before allowing changes.

• It prevents independent binding across layers by using lower layer bindings to validate messages that attempt to change bindings at higher layers.

• It protects against readily changed, but supposedly unique, identifiers by leveraging IEEE 802.1x to provide a root-of-trust for network identifiers, binding the MAC address to a cryptographic authentication and eliminating disconnected host race conditions.

Assumptions. SECUREBINDER assumes that the switches and the controller are not compromised and that OpenFlow messages are cryptographically protected (*e.g.,* with TLS). It also assumes that the controller implements secure topology detection, to be able to correctly differentiate network edge ports from internal ports. This is in line with protection already provided by solutions such as TopoGuard [24].

At a high-level, SECUREBINDER consists of a binding protocol mediator module, a binding store database, a port control module, and a device authenticator module. The binding store maintains authenticated bindings at all layers that the protocol mediator uses to validate binding protocol messages. It is updated by the binding protocol mediator as bindings are updated. The binding protocol mediator itself is responsible for verifying the bindings in these protocol messages, performing additional validation for binding updates, and ensuring that bindings are consistent with lower layers. The port control module is responsible for configuring flow rules on individual network ports to separate binding protocol traffic and enable egress filtering based on identifier binding updates, 802.1x authentication, and changes in port or switch status. The device authenticator is responsible for authenticating the MAC addresses of hosts using 802.1x.

Mediator. The mediator separates identifier binding control traffic, like DHCP, ARP, and 802.1x, from normal data plane traffic and sends it to the control plane. This means broadcast traffic no longer goes to all hosts on the network, enabling all hosts to influence identifier bindings, but only to the controller and a few select applications processing those broadcast requests. Once this identifier binding control traffic reaches the controller, the binding mediator validates it by using the global view of the network enabled by SDN to check incoming binding control traffic against existing bindings. If an attempt is made to rebind an identifier that is already bound, for example, binding an IP address to a different MAC address, the mediator performs additional verification, by checking that the old identifier is no longer reachable, before allowing this rebinding. Similarly, the mediator enforces cross-layer consistency in identifier bindings, requiring binding requests to originate at the same location as the known identifier.

Port control. This module addresses bindings (*i.e.,* MAC address to network location) that are implicitly inferred from network traffic without an explicit signaling protocol. It performs dynamic egress (*i.e.,* source-address) filtering on a per-port basis based on the binding information, thus preventing spoofed packets at the first SDN-controlled port and changing this implicit binding to an explicit one controlled by the configuration of the egress filters. While egress filtering has been used in the past [29], SDN's ability to automatically identify network edge ports and dynamically configure flow rules allows egress filtering to be done automatically.

Device authenticator. While the mediator and port controller securely bind all higher identifiers to a MAC address, they cannot guarantee that a MAC address corresponds to a particular physical device. This is because MAC addresses can be easily changed on all modern NICs and operating systems. This fundamental weakness affects many traditional and SDN-based security and access control systems, including Ethane [6] and techniques that tie a MAC address to a single physical port. The device authenticator addresses this issue by extending IEEE 802.1x, a network access technology supported by all major operating systems and platforms that is designed to enable a network port if-and-only-if an authorized client is connected. Traditionally, the RADIUS authentication server used with 802.1x only verifies that the device is authorized to connect to the network without checking its MAC address or any other network identifier. If the device is authorized, 802.1x enables the port and the device to send arbitrary traffic into the network. We extend the authentication to validate each device's MAC address as well, providing a cryptographic root-of-trust for our network identifier bindings.

802.1x operates using an authenticator on the switch which tunnels EAP [1] messages between a supplicant on the client and a RADIUS [42] authentication server on the backend. The use of EAP and RADIUS enable many authentication mechanisms, including both password based mechanisms (*e.g.,* EAP-MSCHAPv2) and certificate based mechanisms (*e.g.,* EAP-TLS). We choose to deploy EAP-TLS [47], which uses client certificates signed by a CA in the RADIUS server. A client that is able to present a certificate signed by the CA is considered authorized to access the network.

We augment 802.1x to validate the client's MAC ad-

dress by having the (trusted) 802.1x authenticator, implemented in the SDN controller, pass the client's MAC address to the RADIUS server. We then maintain a database in our RADIUS server that associates each certificate's common name with its MAC address.[2] When a new device is added to the network, the admin generates a certificate for it and adds its common name and the device's MAC address to this database. When the client presents its certificate, it is first verified, and then, if it is valid, its common name and the client's MAC are checked against this database. Only if they match is the client authenticated. Note that this database is not accessible outside of the system running the RADIUS server and is only modified manually by the admin.

While we seek to provide a secure network that completely prevents identifier binding attacks, we also take into account the currently-existing network devices. In particular, while 802.1x is supported by the majority of devices and operating systems, it is not yet universally supported. For those devices, like printers or IP phones, that do not support it, SECUREBINDER provides a weak authentication based on the device's MAC address. It is important to understand that this drastically weakens the authentication guarantees provided for that device, allowing an attacker to impersonate that device to the network.

Due to space constraints, for further details we refer the reader to Appendix B.

4.2 Implementation

The introduction of multiple flow tables in OpenFlow 1.3 [38] eases the implementation of SECUREBINDER. In particular, we reserve the first table, table 0, for separating identifier binding traffic from regular data-plane traffic and doing egress filtering, while tables 1+ are used for routing and other applications as normal. High priority flow rules are inserted into table 0 such that all 802.1x, ARP, and DHCP traffic is sent to the controller while DNS and Active Directory traffic are routed directly to their respective servers. Egress filtering is accomplished by inserting flow rules into table 0 such that flows with expected source identifiers (both MAC and IP addresses) are sent directly to table 1 to be routed as normal, while all other traffic is rate limited and sent to the controller.

SECUREBINDER takes the form of a privileged SDN controller application which has configured itself to handle all `packet_in` events before any other application. It then looks for packets sent to it as a result of rules in table 0. Any identifier binding traffic is validated, used to update binding information, and sent to the relevant application. Any other packets sent to the controller from rules in table 0 will be logged and dropped.

We implemented SECUREBINDER as an SDN application in ONOS 1.5.1.[3] We had to make a few modifications, totaling 548 lines of code, to the core of ONOS. The major change, totaling 438 lines, was to secure the implementation of topology detection provided by ONOS. SECUREBINDER leverages secure topology detection as a major performance optimization to dramatically reduce the overhead by only validating packets and installing egress filters at the edge of the network. While existing work like TopoGuard [24] has demonstrated the importance of secure topology detection and provided an implementation, ONOS has not yet incorporated this feature. Secure topology detection protects against a wide range of attacks so we believe it should be a service provided by all SDN controllers. Hence, we use it as an optimization in SECUREBINDER. The remaining modifications reserve table 0 for SECUREBINDER, moving all other applications to table 1.

Our SECUREBINDER application itself is 2,350 lines of Java. Since it processes packets prior to any other application in the controller, it protects other applications in use from incorrect binding information. This enables us to use the existing ONOS ProxyARP and DHCP applications without modification.

5 Evaluation

In this section we provide a formal evaluation of the security provided by SECUREBINDER against identifier binding attacks and evaluate its effectiveness against our new Persona Hijacking attack and its performance impact in a testbed environment.

5.1 Formal Evaluation

In order to assess the security properties of SECUREBINDER, we conducted a formal, model checking-based analysis of hosts interacting via ARP and DHCP over an SDN, both with and without SECUREBINDER in place at the controller. We defined a set of security invariants which, if violated, correspond to the successful malicious use of ARP, DHCP, or IP/MAC spoofing. Using the SPIN model checker [22], we first ran an analysis without SECUREBINDER. This returned a large set of automatically discovered counter-examples (invariant violations) that correspond to known ARP spoofing, Host-Location Hijacking (also independently and manually discovered by Hong, *et al.* [24]), and rogue DHCP attacks, as well as our own IP takeover and Flow Poisoning attacks. Next, we enabled SECUREBINDER in the model and re-ran the analysis against the same set of security invariants. In

[2] This indirection enables a single certificate to be used on a multi-homed host that has multiple MAC addresses.

[3] `http://onosproject.org/`

this case, SPIN was unable to find any security violations, indicating that our defense prevented all of the previously discovered attacks.

Model Checking. Our formal models were written in the Promela process modeling language, and the security properties were checked using the SPIN model checker. Models written in Promela define properties written in Linear Temporal Logic (LTL). LTL has been used to check safety and liveness properties, but also security invariants. For a full explanation of LTL syntax and semantics, the interested reader can refer to the book by Reeves and Clarke [41]. Once a Promela model is written and the logical properties of its state variables defined, SPIN can then be used to verify that those properties hold over all reachable system states, or to find a counter-example (*i.e.,* an attack violating a security property). SPIN models all possible inter-leavings of non-atomic actions in a concurrent system of communicating processes.

Model Architecture. Our analysis considers end-hosts communicating via packets passed to an SDN switch managed by a controller that uses the source address fields of `packet_in` events to populate a NIB used to make routing decisions. A packet consists of an Ethernet frame (containing source and destination MAC addresses) encapsulating either an ARP message or an IP header and DHCP payload.

ARP is implemented as defined in RFC 826 [40], but does not include the message fields or associated checks for hardware and protocol types, since our analysis is focused on IPv4 Ethernet networks. ARP clients may also send gratuitous ARP requests or replies.

A fragment of DHCP is implemented as defined in RFC 2131 [15]. The full client and server state machines are implemented (using symbolic addresses), but we do not include any generic representation of DHCP options or other configuration parameters not essential to the assignment of IP addresses. This is because we define *any* DHCP payload from an unauthorized DHCP server to be malicious, regardless of its content.

Communication occurs through uni-directional channels, in which senders place packets on a finite-length FIFO queue and receivers remove messages from the head of that queue. All communication between end-hosts is mediated by the switch: that is, end-hosts place packets in a switch queue, and the switch determines the end-host queue to which it should forward the packet.

End-hosts are processes which non-deterministically[4] send and receive ARP and DHCP client messages. The target of unicast traffic is chosen non-deterministically. One end-host is also designated as a DHCP server and implements a DHCP server that uses ARP probes to determine if a previously-used address is still in use. All end-hosts faithfully follow protocol specifications (*e.g.,* using correct source addresses).

An adversarial end-host does not follow protocol specifications. Any data field in an Ethernet, ARP, IP, or DHCP message may be non-deterministically assigned any symbolic value (*e.g.,* another end-host's source address). Adversarial end-hosts may also act simultaneously as both DHCP servers and clients.

Security Properties. Our analysis is based on checking the same set of invariants in two different cases: a basic SDN, and one in which SECUREBINDER is deployed. These invariants are designed to completely capture correct network identifier binding behavior in the case of an IPv4 Ethernet network with at most one IP address assigned to each network interface, one end-host connected to each switch port, and no multi-homed end-hosts, as described in Section 2.1. They utilize several ground-truth tracking tables maintained in the model to check the actual values of client and controller data structures against their intended values in the absence of adversarial behavior. The ground-truth invariants are intended to represent 'idealized' network security enforcement, which, while not implementable in the real world, will be violated in the presence of any kind of attack against network identifier bindings. Note that a number of sub-invariants (such as a one-to-one binding of IP-to-MAC and Mac-to-Port mappings) are implicitly captured, as a violation of these would result in a violation of one or more of the explicitly checked invariants.

We define two kinds of security properties: invariants, which must hold in all model states, and assertions, which must hold in a subset of model states. The former are encoded as LTL formulas over model state variables (*e.g.,* the entries in each end-host's ARP cache). SPIN uses an automata-theoretic construction (see [22]) to ensure that no LTL violations occur in any reachable model state, and to return an execution trace in the case that a violation was found. Assertions are encoded as Boolean predicates inserted inline in the model code, which are checked whenever the model executes that line. LTL invariants were used when security requirements constrained the value of a persistent data structure, such as ARP or DHCP tables. The `G` operator in LTL states that the formula must hold Globally over all reachable states. Assertions were used for constraints on the value of non-persistent messages passed over the network.

The security requirements that we checked are presented in Table 2 and discussed in detail in Appendix C.

Results. Because model checking is limited to a finite state space search, it is necessary to bound the size of the network for each analysis. We checked our invariants for networks with a single attacker, DHCP server, and SDN switch. Analyses were conducted for networks with end-host populations ranging from 1 to 20.

[4]Promela's control structures are non-deterministic, because SPIN considers all possible orderings of events in a system.

Table 2: Identifier Correctness Requirements

Requirement Name	Requirement Formula (LTL Invariants and Message Assertions)
Port–MAC Binding	`G(port_to_mac[p] == g_mac_at[p])` for each port `p`
MAC–IP Binding (ARP)	`G(arp_tbls[c][i] == NO_ENTRY || g_mac_to_ip[arp_tbls[c][i]] == i || g_mac_to_ip[arp_tbls[c][i]] == NO_ENTRY)` for each ARP table `c` and each entry `i`
Authorized DHCP	`(msg.cid == DHCP_SERVER && (msg.dhcp.type==DHCP_OFFER || msg.dhcp.type==DHCP_ACK || msg.dhcp.type==DHCP_NAK)) || (msg.cid != DHCP_SERVER && (msg.dhcp.type==DHCP_DISCOVER || msg.dhcp.type==DHCP_REQUEST || msg.dhcp.type==DHCP_DECLINE || msg.dhcp.type == DHCP_RELEASE))`
Genuine chaddr	`msg.dhcp.chaddr == g_mac[msg.cid]`
Genuine ciaddr	`msg.dhcp.ciaddr == g_ip[msg.cid]`
Genuine MAC	`msg.frame.eth_src == g_mac[msg.cid]`
Genuine IP	`msg.ip.nw_src == g_ip[msg.cid]`

Table 3: Attacks Found Through Invariant Checking

Attack Class	Description	Invariant Violated
ARP Spoof	Gratuitous Request with Victim's SPA and TPA and own SHA	MAC–IP Binding (ARP)
ARP Spoof	Gratuitous Reply with Victim's SPA and own TPA	MAC–IP Binding (ARP)
ARP Spoof	Reply to Request with Victim's SPA and own TPA	MAC–IP Binding (ARP)
Host-Location Hijacking [24]	Ethernet packet with victim's MAC	Port–MAC Binding, Genuine MAC
IP takeover	DHCP_Release with victim's IP	Genuine ciaddr
Flow Poisoning	Ethernet packet to victim with target's MAC	Port–MAC Binding, Genuine MAC
Rogue DHCP	DHCP_OFFER from attacker	Authorized DHCP

When SECUREBINDER was not deployed, many invariant violations were found corresponding to existing attacks. Manual inspection of the execution traces revealed that all of these are either known attacks or correspond to our IP takeover or Flow Poisoning attacks. These are summarized in Table 3.

When SECUREBINDER was enabled, no invariant violations were found. This indicates that the set of SDN-based checks implemented by SECUREBINDER is equivalent to the ideal invariants that can be checked with access to ground truth. Note that formal verification via model checking is sound but incomplete, because it is based on a finite state space search of a larger, potentially infinite, space. Model checking can be made complete, however, if it can be shown that the larger region reduces (*e.g.,* via equivalence classes) to the explored region.

We argue (but do not formally prove) that this is the case for our analysis. Above an end-host population of 3, all invariant violations were variants of those already found in networks of 3 or fewer end-hosts. That is, while the specific details of the violation (*e.g.,* the protocol address, hardware address, or ARP table) varied, the actual violating condition (*e.g.,* an IP address bound to a MAC address not assigned by the DHCP server) was one already seen in the smaller analyses. Given this empirical evidence, we suspect that no new attacks will be found by adding more end-hosts to the analysis, nor will any attacks be found in the case that SECUREBINDER is deployed. Clearly, making the analysis more complex in other ways (*e.g.,* allowing multi-homing or multiple end-hosts to share a switch port) may introduce new vulnerabilities. Exploring these more complex scenarios is a component of our future work.

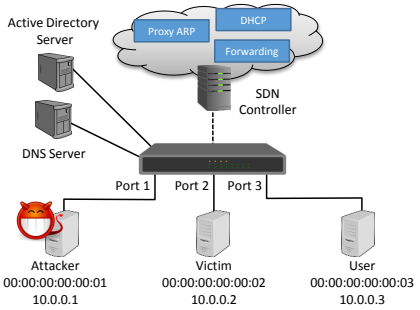

Figure 5: Testbed Evaluation Network Topology

5.2 Experimental Evaluation

We construct an emulated SDN testbed network and launch three separate identifier binding attacks, including our Persona Hijacking attack, at ONOS 1.5.1 with and without SECUREBINDER. To guarantee representative results, we repeat each of the attacks ten times.

For our emulated SDN testbed we used Mininet 2.2.1 [30] with Open vSwitch 2.4.0[5] software switches. We chose a minimal topology with a single switch and

[5] `http://openvswitch.org/`

Table 4: Performance Results

Controller	Host Join	New Flow	pkt_in's
ONOS 1.5.1	505±578ms	8±4ms	131±2
SECUREBINDER	3,505±678ms	6±5ms	193±8
Overhead	+3000ms	-2ms	+62

three hosts—an attacker, a victim, and a user that wishes to contact the victim—for our network, as shown in Figure 5. The attacks we test are also relevant for more complex topologies; we demonstrate them on a minimal topology for simplicity. The experiments were performed in an Ubuntu 14.04.4 VM with 2 cores of an 2.70GHz Intel i7 CPU available and 15GB of RAM. Our test network uses ONOS 1.5.1 as the controller, providing shortest path routing, proxyARP, and DHCP.

Persona Hijacking attack. Since we test against ONOS, we need only the IP takeover phase of the attack. The attack was successful, allowing the attacker to steal the victim's IP address. On average, an attack took 49.8 seconds to execute, with effects lasting indefinitely.

ARP poisoning. The attack was successful, allowing the attacker to receive traffic destined for the victim. This attack lasts until the victim sends traffic that traverses the controller, 41 seconds in our experiments, but depends highly on the workload of the victim machine.

Host location hijacking. In this attack, previously reported by Hong, *et al.* [24], the attacker sends spoofed packets that contain the victim's MAC address as their Ethernet source address with the goal of confusing the SDN controller into thinking that the victim has moved to the attacker's location. If this can be accomplished, traffic for the victim will be sent to the attacker's location. We observed this attack to be completely successful as well, allowing the attacker to receive traffic destined for the victim. Like the ARP poisoning attack, this attack has a limited lifetime. Once the victim sends traffic that traverses the controller, the controller is able to correct the victim's location, ending the attack. In our experiments, this was an average of 51 seconds, but highly depends on the workload of the victim.

In all of the above scenarios, SECUREBINDER threw an alert and blocked the attack immediately.

5.3 Performance Evaluation

We evaluate the additional overhead our defense imposes in terms of extra latency for devices joining the network and on each new flow, as well as the additional controller load and flow rules it generates. We run each experiment 10 times and present averages and standard deviations.

Latency. We measure Host Join Latency and New Flow Latency. Host Join latency measures the latency for a host to join the network and includes network link detection, DHCP negotiation, 802.1x authentication–for SECUREBINDER–, host detection, and flow rule setup and installation of the first flow. New Flow Latency measures the latency to start a new flow—sending a `packet_in` to the controller, forwarding, and rule installation. We measure Host Join Latency from the first packet a host sends until the insertion of the first flow rule. For New Flow Latency, we measure it from the moment the first packet of the flow arrives at the switch until the first flow rule is inserted. No packets after the first in each flow will be diverted to the controller, so any additional latency only impacts the first packet of a flow.

We compared unmodified ONOS 1.5.1, providing shortest path routing, proxyARP, and DHCP, with SECUREBINDER in a network topology with a single switch. Table 4 shows the results. Host Join Latency is higher for SECUREBINDER, at about 3.5 seconds. This is compared to about 0.5 seconds for ONOS 1.5.1. Most of this difference is due to the 802.1x authentication and additional flow rule insertions required by SECUREBINDER. However, 3.5 seconds is actually fairly reasonable considering that Host Join Latency represents the latency for a host to join a new network.

New Flow Latency, by contrast, is essentially the same between unmodified ONOS 1.5.1 and SECUREBINDER. Our results even appear to indicate a slight decrease when using SECUREBINDER, although that difference is within the noise and not actually meaningful.

Controller load. We approximate controller load as the number of packets handled by the controller. While different packets may take noticeably different amounts of processing to handle, this is a common proxy for controller load and does accurately account for the additional load placed on the network via `packet_in` messages, TLS message encryption load, message parsing, and event loop processing.

We measured the number of `packet_in` messages sent to the controller using a Mininet network with 3 switches and 4 hosts in a tree toplogy and compare unmodified ONOS 1.5.1, providing shortest path routing, proxyARP, and DHCP, with SECUREBINDER. Our experiment consists of starting the network, waiting 30 seconds for the network to stablize, performing a pairwise ping between all hosts, and shutting the network down. Our results appear in the third column of Table 4. We observe a 47% increase in the number of `packet_in`'s processed by SECUREBINDER, which is fairly significant. However, of the 62 additional `packet_in`'s, 32 are a result of 802.1x authentication. This means that this additional load occurs only when a new host joins the network.

Number of additional flow rules needed by SECUREBINDER. Flow rules are a limited resource in OpenFlow switches. We can calculate the number of additional flow

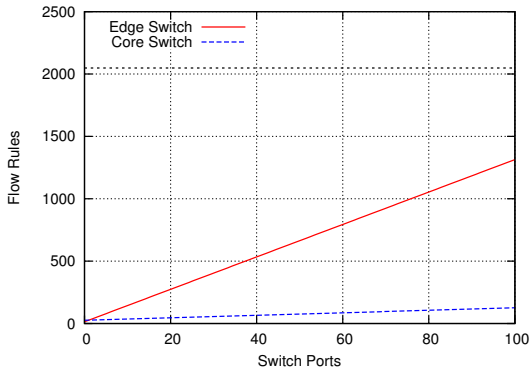

Figure 6: Number of flow rules needed in each switch for SECUREBINDER, as a function of the number of switch ports. Dashed black line marks minimum TCAM rule slots in a modern SDN switch.

rules required per switch by SECUREBINDER as:

$$26 + 13 * edge\_ports + internal\_ports$$

The first term relates to static flow rules installed globally in each switch to send 802.1x, ARP, and DHCP traffic to the controller and block DNS, mDNS, and Active Directory by default. The second term describes flow rules installed for each edge switch port to enable egress filtering and allow non-spoofed Active Directory and DNS traffic destined for the legitimate servers. The final term includes the flow rules that are inserted in table 0 for internal network ports to send traffic directly to the normal forwarding rules in table 1.

We plot this equation for both edge and core switches with between 1 and 100 ports in Figure 6. We assume edge switches have one port connected to the core network with all other ports connected to edge devices while core switches are connected only to other switches. Keep in mind that most edge switches are usually in the 24-48 port range with the very high degree switches in the core of the network. For a 48 port edge switch, SECUREBINDER would require 638 flow rules.

Determining the number of flow rules supported by modern SDN switches is surprisingly challenging. The number of TCAM slots in current SDN switches of about 48-ports varies from around 512 to 8,192 [12, 43, 7, 21] and many vendors claim to be able to support 65,536 flow rules or more [7, 21]. We use 2,048 TCAM entries (available on many switches) as a lowerbound and denote it with a black dashed lined in Figure 6. For this lowerbound, SECUREBINDER would require 31% of the rules in a 48 port edge switch, 16% in a 24 port edge switch, and only 4% in a 48 port core switch. For edge switches, this is a significant, but still practical, overhead; assuming the devices connected to such a switch communicate evenly, each can talk to 29 other devices simultaneously before exceeding the flow table limits. For core switches, the overhead of SECUREBINDER is insignificant. Further, if we consider the use of higher-end switches with 8,192 rules per switch, these overhead figures become 8%, 4%, and 0.9%, respectively.

6 Limitations and Discussion

Although the Persona Hijacking attack is extremely powerful, it does have some limitations as well as several possible partial mitigations that may not prevent the attack but can alert an attentive defender.

DHCP. Principle among these is that the target must be using DHCP for Persona Hijacking to be applicable. Another limitation is that DHCP starvation, despite being extremely transient, is easily detectable and likely to be monitored because it can also indicate network malfunction. In a similar way, the large number of DHCP_DISCOVERs needed to launch the attack would be readily noticed by a network anomaly detector. If such monitoring systems are deployed, Persona Hijacking would quickly be brought to the attention of the network administrators. However, even if detected, mitigation would require the involvement of a human operator, probably on a time scale of tens of minutes to hours.

Another factor that can complicate a Persona Hijacking attack is the use of static DHCP leases fixing IP addresses to specific MAC addresses. If the network protects against MAC spoofing, this will completely prevent Persona Hijacking. However, we are unaware of any SDN controller implementing any form of MAC spoofing protection. As a result, the attacker is free to launch the Persona Hijacking attack by spoofing DHCP_DISCOVERs with the target's MAC address at a different network location. Interestingly, in this case DHCP starvation is not required.

A number of security features commonly found in traditional switches (*e.g.,* port security [9] and DHCP snooping [8]) make it more difficult to launch a DHCP starvation attack by limiting the number of source MAC addresses originating from a single port; however, a recent starvation technique has been developed to bypass these defensive mechanisms [50]. This technique exploits the DHCP server's IP address conflict detection by answering all the probes used to check if an address is in use, *without* spoofed MAC addresses.

Despite these limitations Persona Hijacking remains a powerful attack that co-opts the network infrastructure to propagate a malicious identifier binding that will reliably last for a significant time even in the presence of a vigilant system administrator running many monitoring tools. This level of persistence is unmatched among existing network identifier attacks and gives the attacker a reasonable window in which to achieve their goals. Fur-

ther, while various security features can make this attack more challenging to launch, they do not prevent it.

Our new defense, SECUREBINDER, is designed to prevent not only Persona Hijacking attacks, but also any other identifier binding attack. Much of the existing work on defenses has focused on preventing single attacks. Once such a defense is deployed, the attacker modifies their attack slightly or transitions to a new binding and continues. SECUREBINDER's goal is to end this game of whack-a-mole by providing a defense against all identifier binding attacks. This influenced our design choices.

Use of 802.11 IEEE 802.1x is not required to defend against Persona Hijacking or ARP poisoning. However, it is still an essential component of a unified defense against identifier binding attacks, despite potentially complicated configuration and deployment. In particular, 802.1x prevents MAC spoofing from being used to bypass network access controls, like firewalls. Without 802.1x, an attacker can present a fake MAC address belonging to a more privileged device. This presents network access control systems with a confused deputy problem; a device is identified by its MAC address and any identifiers bound to that address, while the attacker presents a MAC address corresponding to an authorized device. Restricting MAC addresses to specific network ports or ensuring that a MAC address is only present at a single network port at a time can only partially mitigate this attack. There will still be some important system that must be widely mobile and occasionally powered off. Attackers can simply attack this system.

Universal IEEE 802.1x deployment is not needed on networks implementing a Bring Your Own Device (BYOD) or public access policy. The purpose of 802.1x is to prevent the attacker from impersonating a more privileged device. In these networks, all BYOD or public devices are unknown to the network and therefore equally (un)privileged. Hence, no confused deputy problem arises that would require strong device identities. Note, however, that any known and trusted devices sharing the same network *should* use 802.1x to protect themselves from being impersonated.

Applicability to wireless networks. Wireless devices, like cell phones and laptops, are particularly vulnerable to impersonation in this manner. We have focused in this work on wired networks; however, SECUREBINDER is equally applicable to wireless networks. Unfortunately, OpenFlow support for wireless networks is still nascent. While a few efforts have looked at the changes to OpenFlow needed to support wireless networks [34, 54, 13], no code, devices, or emulators exist yet. Note that deploying SECUREBINDER at the first wired switch after a wireless access point would still provide significantly improved security and would prevent attacks that have to traverse the wired network.

7 Related Work

The closest work to ours is Ethane [6], which leveraged the SDN-provided global view of the network to enable access control based on identifiers like hostnames and users. However, it does not provide a root-of-trust for network identifiers, instead authenticating based on MAC addresses, and does not appear to distinguish between creating new bindings and updating existing bindings. Additionally, Ethane does not manage or protect the hostname-to-IP or user-to-hostname bindings.

A number of efforts [23, 20, 25] have investigated access control in SDN networks. This is an important, but orthogonal line of research. Access control allows or denies network flows based on particular network identifiers or characteristics. In this work, we attack and secure the bindings between these identifiers. By breaking these bindings an attacker can gain access to a false identity and all the network access rights of that false identity. Access control is also being applied in the controller to protect against malicious SDN applications [39].

TopoGuard [24] and SPHINX [14] studied attacks on the MAC address to network location binding, which they refer to as Host Location Hijacking. TopoGuard proposes a defense based on differentiating between creating new bindings and updating existing bindings, requiring a host to not be reachable at its old location before updating the binding. SPHINX defends against these attacks by ensuring that new flows conform to existing identifier bindings, preventing spoofed packets. Both defenses are vulnerable to MAC address spoofing.

In traditional networks, several network identifier attacks and defenses have been developed over the years; they tend to only address a single layer of the network stack at a time, and the defenses may only be heuristic in nature. Port Security [9] is a heuristic defense against MAC spoofing, which limits the number of MAC addresses that can be present on a single network port.

To prevent ARP spoofing [11], a wide range of defenses based on replacing ARP with secure variants have been proposed [4, 32, 35]; however, vendor-supported technologies such as Cisco Dynamic ARP Inspection (DAI) [29], which compares ARP replies with DHCP server records, or monitoring tools like `arpwatch` [31] are used more often in practice. These technologies have a number of limitations, including not protecting static IP addresses and requiring manual configuration.

To prevent rogue DHCP servers [49], DHCP Snooping [8] can be used to separate the switch ports into trusted and untrusted zones. This defense requires manual configuration of the trusted and untrusted zones and is limited to protecting against attacks on DHCP only.

Defenses against DNS spoofing [48] include increasing the randomness in the DNS query, using random

source ports and transaction IDs, to protect against blind attackers [48], as well as cryptographic techniques like DNSSEC [2] that protect the authoritative response from tampering. DNSSEC has yet to be widely deployed.

The username to hostname binding can be protected using Kerberos [37] for authentication, as is the case in Microsoft Active Directory, but architectural and implementation issues enable various attacks, such as pass-the-hash [16, 17], in practice.

8 Conclusion

We have built a proof-of-concept attack in SDNs to hijack MAC and IP addresses, steal hostnames, and poison flows to remove victim bindings and accessibility. We have thereafter shown how to use SDN capabilities to prevent such attacks by implementing a new defense that exploits SDN's data and control plane separation, programmability, and centralized control to protect network identifier bindings, and builds upon the IEEE 802.1x standard to establish a cryptographic root-of-trust. Evaluation shows that our defense formally and experimentally prevents identifier binding attacks with little additional burden or overhead.

Acknowledgements

We thank William Streilein and James Landry for their support of this work as well as our shepherd, Guofei Gu, and anonymous reviewers for their helpful comments on this paper. This material is based upon work partially supported by the National Science Foundation under Grant No. 1600266.

References

[1] ABOBA, B., BLUNK, L., VOLLBRECHT, J., CARLSON, J., AND LEVKOWETZ, H. Extensible authentication protocol (EAP). RFC 3748, 2004.

[2] ARENDS, R., AUSTEIN, R., LARSON, M., MASSEY, D., AND ROSE, S. DNS security introduction and requirements. RFC 4033, 2005.

[3] BIGFIX CLIENT COMPLIANCE. Cisco NAC. *BigFix, Inc. 25* (2005).

[4] BRUSCHI, D., ORNAGHI, A., AND ROSTI, E. S-ARP: a secure address resolution protocol. In *ACSAC* (Dec 2003), pp. 66–74.

[5] CANINI, M., VENZANO, D., PERESINI, P., KOSTIC, D., REXFORD, J., ET AL. A NICE way to test openflow applications. In *NSDI* (2012).

[6] CASADO, M., FREEDMAN, M. J., PETTIT, J., LUO, J., MCKEOWN, N., AND SHENKER, S. Ethane: taking control of the enterprise. In *ACM Computer Communication Review* (2007), vol. 37, ACM.

[7] CENTEC NETWORKS. Centec networks - SDN/OpenFlow switch - v330, 2017. `http://www.centecnetworks.com/en/SolutionList.asp?ID=42`.

[8] CISCO SYSTEMS, INC. Catalyst 6500 release 12.2sx software configuration guide, chapter: Dhcp snooping, 2013.

[9] CISCO SYSTEMS, INC. Catalyst 6500 release 12.2sx software configuration guide, chapter: Port security, 2013.

[10] CISCO SYSTEMS, INC. *Cisco IOS Software Configuration Guide*. Cisco Systems, 2013.

[11] DE VIVO, M., DE VIVO, G. O., AND ISERN, G. Internet security attacks at the basic levels. *ACM Operating Systems Review 32*, 2 (1998).

[12] DELL, INC. Dell openflow deployment and user guide 3.0, 2015. `http://topics-cdn.dell.com/pdf/force10-sw-defined-ntw_Deployment%20Guide3_en-us.pdf`.

[13] DELY, P., KASSLER, A., AND BAYER, N. Openflow for wireless mesh networks. In *2011 Proceedings of 20th International Conference on Computer Communications and Networks (ICCCN)* (July 2011), pp. 1–6.

[14] DHAWAN, M., PODDAR, R., MAHAJAN, K., AND MANN, V. SPHINX: Detecting security attacks in software-defined networks. In *NDSS* (2015).

[15] DROMS, R. Dynamic host configuration protocol. RFC 2131 (Draft Standard), Mar. 1997.

[16] DUCKWALL, A., AND CAMPBELL, C. Still passing the hash 15 years later... In *BlackHat* (2012).

[17] DUCKWALL, S., AND DELPY, B. Abusing microsoft kerberos: Sorry you guys don't get it. *BlackHat* (2014).

[18] EHRENKRANZ, T., AND LI, J. On the state of IP spoofing defense. *ACM Transactions on Internet Technology (TOIT) 9*, 2 (2009), 6.

[19] FEKAY, A. AD & Dynamic DNS updates registration rules of engagement, 2012. `http://blogs.msmvps.com/acefekay/2012/11/19/ad-dynamic-dns-updates-registration-rules-of-engagement/`.

[20] HAN, W., HU, H., ZHAO, Z., DOUPÉ, A., AHN, G.-J., WANG, K.-C., AND DENG, J. State-aware network access management for software-defined networks. In *Proceedings of the 21st ACM on Symposium on Access Control Models and Technologies* (2016), SACMAT '16, ACM, pp. 1–11.

[21] HEWLETT-PACKARD DEVELOPMENT COMPANY, L.P. HP switch software OpenFlow v1.3 administrator guide K/KA/WB 15.17, 2015. `http://h20566.www2.hpe.com/hpsc/doc/public/display?sp4ts.oid=5354494&docLocale=en_US&docId=emr_na-c04656675`.

[22] HOLZMANN, G. J. *The SPIN model checker: Primer and reference manual*, vol. 1003. Addison-Wesley Reading, 2004.

[23] HONG, S., BAYKOV, R., XU, L., NADIMPALLI, S., AND GU, G. Towards SDN-defined programmable BYOD (bring your own device) security. In *NDSS'16* (2016).

[24] HONG, S., XU, L., WANG, H., AND GU, G. Poisoning network visibility in software-defined networks: New attacks and countermeasures. In *NDSS* (2015).

[25] HU, H., HAN, W., AHN, G.-J., AND ZHAO, Z. FLOWGUARD: Building robust firewalls for software-defined networks. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking* (2014), HotSDN '14, ACM, pp. 97–102.

[26] KATTA, N. P., REXFORD, J., AND WALKER, D. Logic programming for software-defined networks. In *XLDI* (2012).

[27] KAZEMIAN, P., CHAN, M., ZENG, H., VARGHESE, G., MCKEOWN, N., AND WHYTE, S. Real time network policy checking using header space analysis. In *NSDI* (2013), pp. 99–111.

[28] KHURSHID, A., ZOU, X., ZHOU, W., CAESAR, M., AND GODFREY, P. B. VeriFlow: Verifying network-wide invariants in real time. In *NSDI* (2013).

[29] KING, J., AND LAUERMAN, K. ARP poisoning (man-in-the-middle) attack and mitigation techniques, 2014. `https://www.cisco.com/c/en/us/products/collateral/switches/catalyst-6500-series-switches/white_paper_c11_603839.html`.

[30] LANTZ, B., HELLER, B., AND MCKEOWN, N. A network in a laptop: rapid prototyping for software-defined networks. In *HotNets* (2010).

[31] LBNL'S NETWORK RESEARCH GROUP. arpwatch. Software, 2009. `ftp://ftp.ee.lbl.gov/arpwatch.tar.gz`.

[32] LOOTAH, W., ENCK, W., AND MCDANIEL, P. TARP: Ticket-based address resolution protocol. *Computer Networks 51*, 15 (2007).

[33] MICROSOFT. Dhcp processes and interactions, 2017. `https://technet.microsoft.com/en-us/library/dd183657(v=ws.10).aspx`.

[34] MOURA, H., BESSA, G. V. C., VIEIRA, M. A. M., AND MACEDO, D. F. Ethanol: Software defined networking for 802.11 wireless networks. In *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)* (May 2015), pp. 388–396.

[35] NAM, S. Y., KIM, D., KIM, J., ET AL. Enhanced ARP: preventing ARP poisoning-based man-in-the-middle attacks. *IEEE Comm. Letters 14*, 2 (2010), 187–189.

[36] NELSON, T., FERGUSON, A. D., SCHEER, M. J., AND KRISHNAMURTHI, S. Tierless programming and reasoning for software-defined networks. *NSDI* (2014).

[37] NEUMAN, C., YU, T., HARTMAN, S., AND RAEBURN, K. The Kerberos network authentication service (V5). RFC 4120, 2005.

[38] OPEN NETWORKING FOUNDATION. OpenFlow switch specification, version 1.3.2, 2013.

[39] PADEKAR, H., PARK, Y., HU, H., AND CHANG, S.-Y. Enabling dynamic access control for controller applications in software-defined networks. In *Proceedings of the 21st ACM on Symposium on Access Control Models and Technologies* (2016), SACMAT '16, ACM, pp. 51–61.

[40] PLUMMER, D. An ethernet address resolution protocol. RFC 826 (Informational), Nov. 1982.

[41] REEVES, S., AND CLARKE, M. *Logic for computer science*. Citeseer, 1990.

[42] RIGNEY, C., WILLENS, S., RUBENS, A., AND SIMPSON, W. Remote authentication dial in user service (RADIUS). RFC 2865, 2000.

[43] SHAMUS MCGILLICUDDY. Pica8 doubles flow rule capacity in its new OpenFlow 1.3 switch, 2014. `searchsdn.techtarget.com/news/2240214709/Pica8-doubles-flow-rule-capacity-in-its-new-OpenFlow-13-switch`.

[44] SHIN, S., AND GU, G. Attacking software-defined networks: A first feasibility study. In *HotSDN* (2013).

[45] SHIN, S., PORRAS, P., YEGNESWARAN, V., AND GU, G. A framework for integrating security services into software-defined networks. *Open Networking Summit* (2013).

[46] SHIN, S., YEGNESWARAN, V., PORRAS, P., AND GU, G. Avant-guard: Scalable and vigilant switch flow management in software-defined networks. In *CCS* (2013).

[47] SIMON, D., ABOBA, B., AND HURST, R. The EAP-TLS authentication protocol. RFC 5216, 2008.

[48] STEINHOFF, U., WIESMAIER, A., AND ARAÚJO, R. The state of the art in DNS spoofing. In *ACNS* (2006).

[49] THE CISCO LEARNING NETWORK. Spoofing attacks: Dhcp server spoofing, 2014. `https://learningnetwork.cisco.com/docs/DOC-24355`.

[50] TRIPATHI, N., AND HUBBALLI, N. Exploiting dhcp server-side ip address conflict detection: A dhcp starvation attack. In *Advanced Networks and Telecommuncations Systems (ANTS), 2015 IEEE International Conference on* (2015), IEEE, pp. 1–3.

[51] VIXIE, P., THOMSON, S., REKHTER, Y., AND BOUND, J. Dynamic updates in the domain name system (DNS UPDATE). RFC 2136, 1997.

[52] WANG, H., XU, L., AND GU, G. FloodGuard: a DoS attack prevention extension in software-defined networks. In *45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)* (2015), IEEE, pp. 239–250.

[53] WELLINGTON, B. Secure domain name system (DNS) dynamic update. RFC 3007, 2000.

[54] YAP, K.-K., KOBAYASHI, M., SHERWOOD, R., HUANG, T.-Y., CHAN, M., HANDIGOL, N., AND MCKEOWN, N. Openroads: Empowering research in mobile networks. *SIGCOMM Comput. Commun. Rev. 40*, 1 (Jan. 2010), 125–126.

A Network Identifiers

Network protocols rely on identifiers of the communicating entities in order to achieve their goals. Such goals, are not only to ensure delivery of packets from a source to a destination, but also to enforce access control and authorization policies (*e.g.,* authority to update a DNS record or to access a service using Kerberos). The key identifiers used at different layers of the networking stack are: network location, MAC address, IP address, hostname, and username.

We consider the *device* as the basic entity with an identifier, be it an end-host, a server, a printer, or an embedded system, *etc.* Devices are distinct from *users* of those devices. Devices may have multiple network interfaces (*e.g.,* virtualized interfaces or multiple Network Interface Cards (NICs)), which may have different identifiers.

Network Location: The lowest level network identifier is the physical switch and port to which a device is connected. We refer to this identifier as the Network Location of a device and define it as a tuple $(switch, port)$, where *switch* is a unique identifier for a switch, a serial number or management IP address in traditional networks and a Data Path Identifier (DPID) in OpenFlow SDNs, and *port* is an integer representing the port number on that switch. As a device's network location is at the edge of the network, a device could potentially have multiple network locations if it is multi-homed. Additionally, multiple devices may be associated to the same network location due to, *e.g.,* virtualization of end-hosts. Thus the mapping between devices and network location is a many-to-many mapping.

MAC Address: A MAC address identifies a NIC or group of NICs on an Ethernet network. There are three

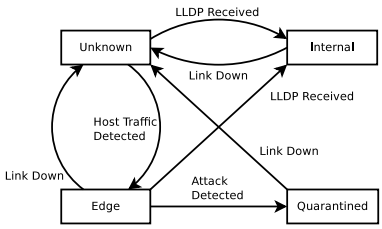

Figure 7: State Machine for each Network Port in SECUREBINDER

types of these addresses: Unicast, Multicast, and Broadcast. Multicast and broadcast addresses allow sending traffic to a particular subset of devices simultaneously while unicast addresses are intended to unambiguously identify a device on the network. Unicast MAC addresses need to be unique in any given Ethernet network or traffic mis-delivery will occur; a unicast MAC address is assigned to each NIC during manufacture.

IP Address: An IP address is used to route traffic across and between networks to a particular device. There are four types of these addresses: Unicast, Multicast, Anycast, and Broadcast. The last three categories are special addresses used to send traffic to a particular subset of devices simultaneously. A Unicast IP address is a unique identifier for a device interface which is constrained to a particular subnetwork (to enable route aggregation). They are either statically configured or assigned using DHCP for IPv4 or DHCPv6 for IPv6. IPv6 also adds a stateless autoconfiguration assignment mechanism known as SLAAC. Only one device in a network can have a particular unicast IP address.

Hostname: A hostname is a human readable name for a system that can be used instead of an IP address. Since one hostname could be associated with multiple IP addresses and one IP address could be associated with multiple hostnames, this is a many-to-many mapping.

Username: A user account identifies a particular user logged onto a system (denoted with a hostname). Many users may be logged into the same system and a single user may be logged into multiple systems, making this a many-to-many mapping.

B SECUREBINDER Design Details

In SECUREBINDER, each network port in the system is in one of four states: `Unknown`, `Internal`, `Edge`, or `Quarantined`. Each port initially comes up in the `Unknown` state, where all traffic is sent to the controller. A port connected to another switch, as identified by the controller's topology detection using LLDP, is put into the `Internal` state, where it sends all traffic directly to table 1 for forwarding. Once at least one host is detected on a port, it is put into the `Edge` state, where it is part of the network edge. In this state, rules are inserted to send packets from known and validated source addresses directly to table 1 while all other traffic is sent to the controller. Finally, a port is placed into the `Quarantined` state when it has been determined that a device on that port is misbehaving. All traffic from `Quarantined` ports is dropped. See Figure 7 for the state machine.

To protect the MAC address to network location binding, we use egress filtering along with 802.1x such that all packets except 802.1x frames that are not associated with an existing binding on a port are dropped. 802.1x frames are passed to our system's 802.1x authenticator in the SDN controller, where the encapsulated EAP messages are sent to the RADIUS server.

We use EAP-TLS authentication, which requires the client to present a valid certificate signed by our internal CA. Additionally, we maintain a database mapping certificate common names to MAC addresses and require the MAC address of the client (as recorded by the trusted 802.1x authenticator in the SDN controller) to match the MAC address associated with the the common name of its certificate in the database. This database is updated manually by the administrator as part of the initial device configuration. If the authentication succeeds, we bind this MAC address to this port and insert flow rules sending packets with this MAC address and port to table 1, for forwarding.

To identify hosts that have disconnected, we listen for port down events and 802.1x log-off messages and remove the corresponding MAC to location bindings. To account for cases where a host may leave the network without sending a log-off message and without the port going down (*e.g.,* a device behind a hub), we periodically query all idle hosts with an ARP ping; devices that do not respond are removed.

We provide support for non-802.1x devices by monitoring the MAC addresses seen on each port and checking whether each one attempts 802.1x authentication within 60 seconds of connecting. If it does not, we assume the device does not support 802.1x and send a RADIUS access request where the username and password attributes simply contain the device's MAC address. The RADIUS server uses a separate database to look up this username and password pair (*i.e.,* MAC address) to see if this device has been granted access. If it has, we then setup the MAC address to network location binding and egress filters to allow it access to the network. Note that since there is no guarantee that this MAC address represents the expected physical device, we recommend statically configuring the network port that each of these devices may connect to and placing stringent ACL rules in the network for such devices to limit network access to only expected locations.

To secure the binding from IP addresses to MAC ad-

dresses, we insert high priority flow rules sending all DHCP and ARP traffic to the controller. Our controller application then checks these packets to ensure that they are self-consistent (*i.e.,* source identifiers in the Ethernet header match those in the ARP/DHCP headers) and that they are consistent with our existing IP to MAC and MAC to location bindings. Inconsistent packets are dropped while validated packets are passed off to an external DHCP server or the controller's ProxyARP application to handle. Note that since our application is the first application to handle these packets, any packets with invalid mappings will be dropped by our application before they can poison other controller applications.

For DHCP, we drop all server messages except those originating from the legitimate server's network location, thereby preventing rogue DHCP servers. We also track the IP address assigned by the server to update our IP to MAC binding information.

We support manually configured static IP addresses by requiring the IP to MAC mapping to be entered in a configuration file. This static IP can be additionally constrained to a single network location. Note that traditional networks with multiple subnets would require similar configuration.

Once we have an IP to MAC address binding, whether from DHCP or static configuration, and know the location of this MAC address in the network, we update the egress filters. In particular, we add one flow rule matching on the port, MAC, and IP address that belongs to this device which sends legitimate traffic to table 1 for forwarding, and we add a second rule that sends all other IP traffic from this port and MAC address to the controller (after a rate limit), preventing IP spoofing.

Interestingly, with 802.1x, DHCP, and manual static IP configuration, we can automatically populate the MAC to location and IP to MAC bindings for all possible reachable hosts. This means we never need to depend on ARP replies from end hosts to populate our bindings. This completely eliminates all ARP poisoning attacks, which operate by either changing the IP to MAC binding or the MAC to location binding.

To secure the hostname to IP address binding, we insert high priority flow rules to drop spoofed DNS packets and send all valid DNS requests to the DNS server while dropping all DNS replies that do not originate at the legitimate DNS server. This prevents the operation of rogue DNS servers and the use of alternate DNS servers. We also drop all multicast DNS and NETBIOS traffic because the broadcast nature of these protocols makes them inherently insecure.

Finally, to secure the username to hostname binding we separate directory service traffic from the dataplane by inserting high priority flow rules to send this traffic directly to the directory server while dropping all spoofed packets. This prevents rogue directory servers and many replay attacks.

C SECUREBINDER Security Requirements

We used SPIN to check our formal model of identifier bindings and SECUREBINDER against the following security requirements listed in Table 2 in Section 5. These formal security requirements attempt to capture the following natural goals:

- **Port–MAC Binding** checks that the SDN's mapping of MAC addresses to switch ports is consistent with the ground-truth mapping.
- **MAC–IP Binding (ARP)** checks that for every entry in the client's ARP Table, one of the following properties holds:
  - There is no MAC address for the associated IP.
  - The MAC address for that IP address is the ground-truth owner of that IP address.
  - There is no ground-truth owner of that IP address. This condition arises due to stale ARP cache entries for a released IP address.
- **Authorized DHCP** checks that DHCP messages which should only be sent by the DHCP server are sent by the DHCP server. It also checks that messages which should only be sent by a DHCP client were not sent by the server. This assertion is checked whenever a DHCP message is received by a client or server, prior to any other packet processing.
- **Genuine chaddr** checks that the client hardware address in a DHCP message matches the ground-truth MAC address of the sender. This assertion is checked whenever the DHCP server receives a DHCP message.
- **Genuine ciaddr** checks that the client network address in a DHCP message matches the ground-truth IP address of the sender. This condition is checked whenever the DHCP Server receives a DHCP_REQUEST or DHCP_RELEASE.
- **Genuine MAC** checks that the source MAC in an Ethernet frame matches the ground-truth MAC address of the originator. This condition is checked whenever a packet is received on a switch port.
- **Genuine IP** checks that the source address in an IP header matches the ground-truth IP address of the originator. This condition is checked whenever a packet is received on a switch port.