No One In The Middle: Enabling Network Access Control Via Transparent Attribution

Jeremy Erickson
University of Michigan
jericks@umich.edu

Qi Alfred Chen
University of Michigan
alfchen@umich.edu

Xiaochen Yu
University of Michigan
ririi@umich.edu

Erinjen Lin
University of Michigan
ejlin@umich.edu

Robert Levy
University of Michigan
roblevy@umich.edu

Z. Morley Mao
University of Michigan
zmao@umich.edu

ABSTRACT

Commodity small networks typically rely on NAT as a perimeter defense, but are susceptible to a variety of well-known attacks, such as ARP spoofing or direct attacks on vulnerable services, by any device already on the network. With the increased prevalence of networked Internet-of-Things (IoT) devices now taking up residence in homes and small businesses, the potential for abuse has never been higher. Despite the prevalence of prior work with similar goals, previous solutions have either been unsuited to non-Enterprise environments or have broken compatibility with existing network functions and therefore have not been adopted. In this work, we present a novel mechanism for strongly attributing network traffic to its originating principal, fully-compatible with existing networks. For the first time, we can apply tried-and-true security mechanisms, such as access control, to legacy network environments where enterprise solutions are often impractical. We eliminate Man-in-the-Middle attacks at both the link and service discovery layers, and enable users to identify and block malicious devices from direct attacks against other endpoints. Our prototype imposes negligible performance overhead, runs on an inexpensive commodity router, and is ready for real-world use *today*.

1 INTRODUCTION

Access control is one of the oldest, most used, and strongest security mechanisms in use today. However, in today's home and small business networks it is difficult, if not impossible, to enforce any meaningful access control in the presence of adversaries. Unlike the local-device permission systems [33], which can enforce access control policies by binding a user's identity to the user's account, the modern network stack is distributed and anonymous. Research on access control in distributed systems has generated important models such as end-to-end encryption [37], ticket-granting [38], and the Public Key Infrastructure [23]. However, these models have historically been implemented at the application layer of the network stack and require coordination between compatible client devices. Due to the difficulty in applying these models to the lowest layers of the network stack while retaining compatibility with legacy devices, our networks are still vulnerable to a variety of well-known attacks today.

Identifiers such as MAC and IP addresses of a device are used by low-level networking protocols to direct traffic, but can be easily changed or spoofed and are not sufficient for strong attribution. The ARP spoofing attack, for instance, conflates the MAC and IP addresses of two different devices on the network to enable an attacker to intercept the network packets of another device, opening the door for data theft and the spread of malware. Well-known attacks like this continue to pose a real threat, especially on home networks and public wireless networks that lack network administrators or the capability for enterprise solutions. Fundamentally, this is caused by the lack of an effective low-layer mechanism to attribute network functions to their originating principals. We identify this as the *Attribution Problem*.

Despite this, commodity small networks, such as home and small business networks, have traditionally been regarded as safe since the router's Network Address Translation (NAT) prevents unauthorized inbound connections from the Internet and devices on the local network are generally benign. However, the Internet of Things (IoT) has dramatically shifted the threat landscape for edge networks. In addition to transient devices, such as an infected laptop brought in by a guest, the widespread security weaknesses in today's IoT devices have been shown to cause large scale infections in home and small business networks [19, 32, 36, 45]. The Vault 7 leaks [44] have revealed weaponized code designed to invade the home network, including an attack that turns Samsung smart TVs into audio recorders. Once the perimeter defense has been bypassed, compromised IoT devices can serve gateways to attack other devices on the network. Devices with valuable content, such as smartphones and laptop computers, are common targets for attacks such as WannaCry [8]. Sharing a local network with adversaries is risky enough that the US Department of Homeland Security cautions users to avoid connecting to public WiFi when possible [26]. Network-based firewalls are ineffective when adversaries can change their identifiers at will. Without the capability for strong attribution, malicious devices have free reign to intercept users' network traffic and launch unfiltered attacks against their targets.

Enterprise network security solutions can defend against these attacks, but are poorly suited to small networks due to cost and complexity. Small network security has therefore been an area of active research. However, previous work has failed to make an impact because: 1) it focused on specific attack patterns and did not generalize to protect against other categories of attacks, 2) it required clients, many of which run proprietary software and cannot easily be updated, to implement new network protocols, 3) it was not fully compatible with legacy use cases, or 4) it was not

easy to use by an only moderately technical user. Most importantly, no previous work has directly addressed the Attribution Problem.

In this paper, we present what we believe to be the first comprehensive solution to the Attribution Problem, focusing first on commodity small networks. Our design provides strong network attribution at the router, which, as the central point in the network, is capable of defending all hosts on the network simultaneously. To provide strong attribution, the router first assigns unique credentials to each device in a network and then binds these credentials to separate virtual network interfaces (vNICs) on the router. This allows the router to effectively identify and differentiate flows between devices so that each network packet can be uniquely associated with its originating principal at the virtual hardware layer. To achieve compatibility with existing devices and network functions, we leverage the credentials used by existing wireless authentication protocols so that the attribution process is completely transparent. This attribution layer is designed to reside below even the link layer of the network stack, thus requiring no changes to existing devices or protocols and cannot be tampered with by an adversary short of compromising the router itself.

Leveraging the support of the strong attribution at such a low layer, we are able to design and implement security modules to defend against local network attacks at any higher layers of the network stack. In this paper, we build several security modules: 1) *Checkpoint*, which eliminates Man-in-the-Middle (MitM) attacks at the link layer, 2) *Dreamcatcher*, which enforces a demand-driven access control policy for devices on the network, and 3) *Warden*, which enables high-level network applications to tailor their functionality to specific devices.

Users primarily interact with Dreamcatcher, a *user-driven* access control system [35], which provides simple interfaces for enrolling new devices on the network and managing the network's access control policy. As devices exercise network functionality, Dreamcatcher will create rules to extend the policy *on-demand*, enabling users to allow or block connections between devices. Our current access control model prioritizes simplicity and usability, as appropriate for a small network environment, although alternate models with finer- or coarser-grained permissions are possible. All traffic is categorized into one of four high-level rule categories and expressed to the user in plain English. We show that this model is approachable for most users and protects against several entire categories of attacks.

We present the following contributions:

- A new mechanism for strong attribution on commodity small networks, fully backwards-compatible with existing protocols, that can protect all layers of the network stack.
- An evaluation of our approach that shows our approach is effective in defending against several categories of network attacks, including ARP spoofing, name poisoning, server-side registration spoofing, and direct attacks. The network performance overhead is also shown to be negligible.
- Data from two usability studies evaluating the efficacy of an attributable network in practice. These studies show that for most users, Dreamcatcher does not interfere with normal network operations and can effectively block the majority of attacks we tested.
- A prototype implementation of our attribution mechanism and new security modules, running on a $50 commodity router, released as open source. In addition to our experimental network environment, our prototype currently sees daily real-world use defending our lead author's home network.

2 CHALLENGES AND RELATED WORK

Challenges. Besides the Attribution Problem, there are two main challenges that related works have difficulty overcoming. The first challenge is **usability**. Commodity small networks differ from wide-area networks and enterprise networks in that they do not have dedicated expert-level network administrative staff to monitor and defend the network. To be practical, any defenses must be either entirely automated or sufficiently simple that extensive technical expertise is not necessary. Along the same lines, complex solutions typically require extensive infrastructure and funding for ongoing support. These are likely impractical for the owners of most small networks.

The second challenge is **compatibility**. New network defenses must remain compatible with existing devices and network functions. Modifications to existing protocols that require client software updates can prove challenging to distribute. For some devices running proprietary software stacks, supporting new protocols can be near impossible. Even solutions that don't require software modifications must support existing network protocols. Solutions that change the way the network operates must continue to support non-standard use cases.

Related work. Demetriou et al. has proposed HanGuard [11], a similar router-based access control platform. HanGuard assigns each IoT application on smart phones a *token*, which allows the application to pass traffic past a monitor service on the router to the IoT device, and other traffic to vulnerable IoT devices will be blocked. This architecture enables the router to differentiate between and filter traffic from different applications within a single device, which is more fine-grained than our approach can currently handle. However, it assumes that malware will be confined to an application and that the smart phone itself is not malicious, and is thus not secure against device-based exploits such as Stagefright [5], dirtyc0w [14] and Drammer [42]. Path authentication, such as ICING [25] and OPT [24], similarly allows routers to determine packet origin and filter traffic that violates the network's security policy. However, all of these techniques require each client device to load additional software, which for many devices does not exist, whereas our approach is platform agnostic and compatible with all modern devices.

In enterprise environments, Intrusion Detection Systems (IDS) and Intrusion Prevention Systems (IPS) are both common and effective. Systems such as Snort [34] and Bro [30] identify network anomalies and alert an administrator of potential attacks, but ultimately the onus rests on the administrator's shoulders to filter false positives and provide remediation for any successful attacks. These systems also typically require dedicated infrastructure to handle the analysis load. While Snort can be run on a commodity router, it relies on up-to-date signatures lists to be effective, which imposes subscription fees on the network owner.

The Sense router by F-Secure [17] is one commercial product that aims to provide an enterprise-level IPS service for the home network. By outsourcing its traffic analysis to the cloud, it avoids the usability challenge of having an expert-level administrator on-call. However, its $199 price tag, $96/year subscription fee, and its reliance on cloud services that may be inaccessible during an outage [7] or be shut down if the product is not a commercial success [16], may make this option unpalatable for many consumers.

Various research studies have focused on defenses against popular techniques for performing Man-in-the-Middle (MitM) attacks on local networks. Two examples are the ARP spoofing and name poisoning attacks. Both Jinhua et al. [22] and Ramachandran et al. [31] proposed techniques that reduce the delay in detecting ARP spoofing through active scanning of hosts on the network to look for inconsistencies. Tripunitara et al. [40] present a heuristic-based approach which generates alerts upon detection. However, these do not prevent ARP spoofing but only make it easier to detect, and do not address the need for human intervention after detection. Ortega et al. [29] proposed a different approach that eliminates the opportunity for ARP spoofing by restricting ARP traffic to the connections between each individual device and the router. The router itself uses its own record of DHCP leases to maintain a mapping of MAC to IP addresses, and forges ARP replies to each client. Unfortunately, this interferes with other standard network operations, such as static IP addressing. Bruschi et al. [9] implemented an authenticated ARP protocol that allows clients to verify the validity of the ARP packets they receive. However, it is limited by its reliance on a Public Key Infrastructure (PKI) to manage and distribute keys among clients, and thus does not work with legacy devices. Similarly, Tian et al. [39] propose an ARP verification protocol necessitating a client TPM, which precludes compatibility with existing software stacks and many devices, IoT devices in particular. Xing et al. identified the capability for name poisoning attacks in the mDNS protocol, and proposed a defense called *speak out your certificate*, which associates the device owner's certificate with her voice to allow biometric-based verification of the device owner's identity [4]. Unfortunately, this solution also requires client devices to update their software to support an extension to mDNS-based service discovery, and does not work with autonomous devices, such as printers and most IoT devices.

Despite the prevalence of such solutions in the literature, commodity small networks are essentially as vulnerable today as they were two decades ago, since no such techniques have been adopted *in practice*. All of these techniques modify the network architecture to deal with specific MitM attacks, but none are easily extended to deal with other categories of attacks, and all of them break backwards compatibility in some way or require unclear user remediation, rendering them inconvenient and preventing their adoption. By comparison, our approach blocks MitM attacks at both the link and service discovery layers simply by associating each network action with its originating principal. MitM attacks, which rely on the network's inability to differentiate between the victim and attacking device, become trivial to block in such an environment. By performing this crucial attribution step below the level of any network protocol, we can avoid interfering with devices' network protocols and retain compatibility with even unconventional use cases. We also show that despite our approach's reliance on user input, it remains effective despite occasional user error and requires only minimal effort primarily over a short period of setup time.

3 THREAT MODEL

In this work, we consider any attack on commodity small networks, even zero-day attacks. We assume that the router is trusted, as our implementation runs on it, but otherwise any device on the network may be compromised. Compromised devices may attempt to send any arbitrary network packets, even maliciously crafted ones, to the router or other devices on the network.

We expect the network administrator to have a moderate technical background sufficient to set up a home or small business router and understand basic networking concepts, such as the fact that transferring a file between computers requires one computer to attempt to *connect* to the other. We make no such assumptions about other users of the network, although we do provide new users with a tutorial on basic network operation and potential attack vectors, modeling how one may teach a friend how to use the system to protect themselves. We assume that our users are *motivated* to learn how the system works and how to use it effectively. While our system is open for everyone, we expect those with some degree of technical proficiency will receive the most benefit.

Overtly malicious behaviors, such as malware which causes a Denial of Service (DoS), are outside the scope of this paper. We assume that they will be detected by the administrator, perhaps by leveraging the attribution mechanism provided in this paper, and that an appropriate remediation (e.g., unplugging the device) will be attempted. Thus, the primary threats we consider are stealthy attacks that aim to subvert the integrity or confidentiality of legitimate communications, or attempt to directly compromise other devices, without visibly affecting normal device operations.

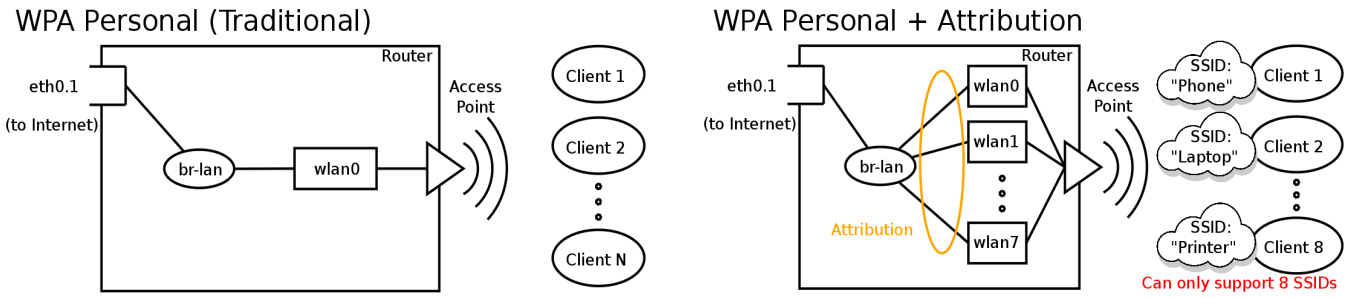
We specifically note that our work does not attempt to completely block all threats, but to *enable* users to more effectively defend their network. Ultimately, the degree to which this is possible depends on many factors including which devices are initially compromised and the user's understanding of the system.

4 METHODOLOGY

The primary contribution of this work is enabling strong attribution on the commodity small network. With this capability, we then demonstrate what an alternate security paradigm on the network may look like. Thus, this section of the paper first focuses on the principles and specific techniques we use to provide attribution, followed by a series of additional security applications that are enabled by strong attribution. This is by no means an exhaustive list, but rather a demonstration of the potential of a network which can strongly authenticate its principals for the first time.

4.1 Attribution mechanism

Local network security has always been a challenging prospect. The device identifiers (MAC and IP addresses) used in all low-level protocols such as Address Resolution Protocol (ARP), switching, and routing are not unique and can be changed or spoofed by any device. The existing computing base depends on these legacy protocols for correct operation, and thus previous attempts to replace these



(a) All devices on the network share a single wireless network and vNIC.

(b) One wireless network is allocated per device. Each wireless network is assigned a vNIC. We can then associate traffic to each device by examining from which vNIC it originates.

Figure 1: Internal network architectures of WPA Personal networks, with and without attribution.

identifiers with more secure alternatives [4, 9] have ultimately failed to be adopted.

Design Overview. Fundamentally, our goal is to split the flows between devices so they can be uniquely identified and differentiated. This is done in two steps. We must first assign unique credentials to each device, and then bind these credentials to an architectural component suitable for use in our security modules. To do this, we leverage the credentials used in existing wireless authentication protocols and associate each device's credentials with its own virtual network interface (vNIC) on the router. By associating each device with a unique vNIC, we make each device's traffic accessible for packet filtering using physical-layer criteria. Operating at such a low layer enables us to defend any higher layer while remaining compatible with all network protocols.

4.1.1 Assigning unique device credentials.
In most small networks using **WPA Personal** network authentication, our credentials consist of the SSID, or name of the network, and a pre-shared key used across all devices that connect to the network. Conceptually, we can tweak this model so the name of the network represents a publicly-visible username, and the pre-shared key represents a unique credential for a *single device*, so long as that credential is not shared between multiple devices. Modern consumer routers often support the ability to run multiple wireless networks using the same physical radio, each with a unique pre-shared key. We can assign unique credentials to each device by creating a separate wireless network for each.

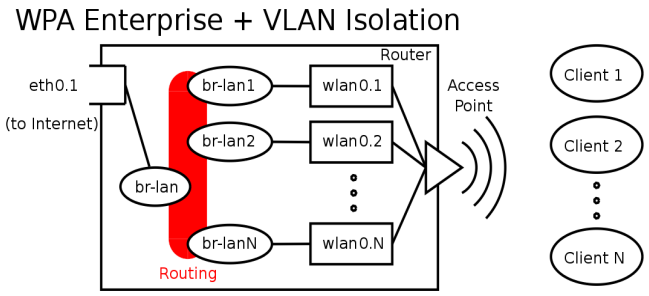
Enterprise environments typically use a different wireless security protocol, often referred to as **WPA Enterprise**. This model, designed to support hundreds or thousands of clients simultaneously, assigns each user their own credentials, typically in the form of a username and password.

These two techniques each have drawbacks. Unfortunately, commodity routers are often limited below the software layer to hosting a maximum of 8 wireless networks per radio. This is not a fundamental limitation of the approach, but rather a practical limitation imposed by router manufacturers. Since our use of WPA Personal networks can only support a single device while maintaining unique credentials for each, we quickly run out of available networks if we only use this technique. WPA Enterprise does not limit the number of devices but is not universally supported. While modern PCs and smartphones universally support WPA Enterprise, some simpler *legacy* devices, such as network printers and streaming media players, may only support WPA Personal. However, by using both WPA Personal and Enterprise authentication mechanisms simultaneously, we can mitigate the downsides of both.

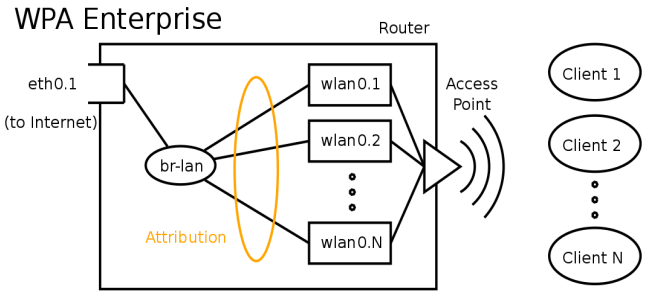
Our approach uses a single main WPA Enterprise network to connect all modern computing devices, and then uses the remaining 7 available wireless networks to support a single legacy device each. Routers which include a second radio on the 5Ghz band (as most do) can host an additional 8 wireless networks, bringing the total number of supported legacy devices to 15. Thus, by using this approach, we can support an effectively unlimited number of modern devices and up to 15 legacy devices simultaneously on our experimental network, each with unique credentials. We believe this reaches the limits of what is possible without manufacturer support, and sufficient for the majority of small network use cases.

4.1.2 Binding device credentials to vNICs.
After assigning a unique credential to each device, we must provide a way for applications to seamlessly attribute flows to devices. For both of the credentialing techniques presented above, we can split each device's traffic out into a separate vNIC within the router based on the credentials provided. As shown in Figure 1b, when we generate multiple SSIDs as is done with the WPA Personal method, each will create its own virtual network interface: `wlan0`, `wlan1`, and so on. These virtual network interfaces, or vNICs, are connected to the main local network bridge and so all connected devices will share the same network.

However, using our WPA Enterprise technique, all devices share the same wireless network, and therefore the same vNIC by default. To split each device's traffic into separate vNICs, we use an enterprise feature: VLAN Isolation. VLAN isolation allows administrators to specify that each wireless client will be placed on a pre-defined virtual local area network (VLAN), isolated from other VLANs. As shown in Figure 2a, we can create a new VLAN with associated vNIC for each device to robustly attribute network traffic. Unfortunately, by itself, this approach interferes with normal network operation. Placing each device on a separate VLAN also puts it on a separate network. Many local network protocols, such as ARP, DHCP, and mDNS, fail to traverse the network boundaries in this configuration. To solve this issue, we modify the *hostapd* utility responsible for generating the virtual network infrastructure to dynamically connect each vNIC to the main LAN bridge as



(a) VLAN Isolation puts each device on a separate network. Unfortunately, this prevents local network traffic from propagating.



(b) We modify the way the router adjusts the virtual network architecture so that VLANs are automatically bridged together. This enables local network traffic propagation while keeping attribution.

Figure 2: Internal network architectures of WPA Enterprise networks, with VLAN Isolation and modified for attribution.

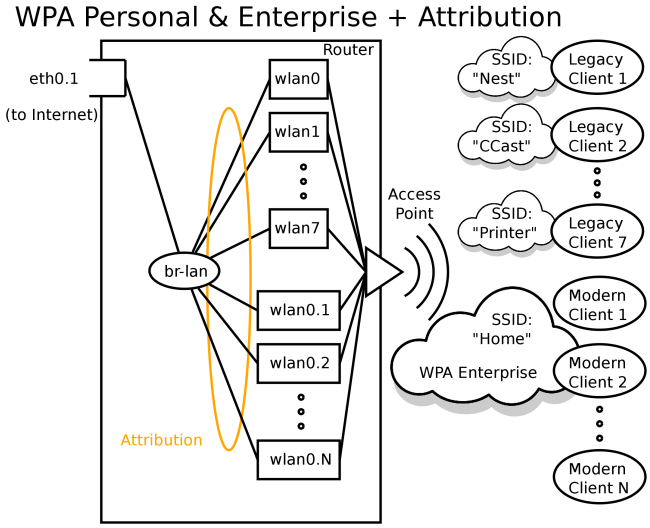


Figure 3: By combining both techniques for attribution on WPA Personal and WPA Enterprise networks, we can support an effectively unlimited number of modern devices and 7 legacy devices, or 15 legacy devices with a second radio.

clients enter and leave the network. This configuration is shown in Figure 2b. Thus, despite being on separate VLANs, all devices share the same network and local network protocols succeed. The architecture for our combined approach supporting both WPA Personal and WPA Enterprise attribution techniques is shown in Figure 3.

4.2 Security applications

Leveraging the support of the strong attribution support, we are able to design and implement security modules to defend against local network attacks at higher layers of the network stack. First, we present **Checkpoint**, a completely transparent solution to ARP and MAC spoofing attacks. Next, we present **Dreamcatcher**, an access control system for the commodity small network, and explore the trade-off space in balancing usability with permission granularity. Finally, we present **Warden**, a module which extends device attribution to user-space applications.

4.2.1 *Checkpoint: link-layer integrity checking.*

In IP/Ethernet networks, the ARP protocol is used to resolve a target IP address to its corresponding MAC address so a packet can be addressed to its destination on the local network. In ARP spoofing attacks, a malicious device conflates the target IP address with its own MAC address so that packets are rerouted to the malicious device.

In traditional networks, this is a difficult problem to solve since MAC addresses cannot be trusted to uniquely identify devices. However, our new attribution mechanism guarantees that each device's traffic will be bound to a unique vNIC, and thus Checkpoint can filter ARP traffic based on a strong device *identity*. To address ARP spoofing, Checkpoint allows each device to *claim* IP addresses as a byproduct of the Sender Protocol Address field (i.e. source IP address) on any sent ARP packet. If an IP address is claimed by a device, Checkpoint will block any ARP packets from other devices claiming that same address. Devices must periodically re-claim IP addresses, or the claim will expire. This simple mechanism stops ARP spoofing attacks entirely.

Checkpoint also prevents MAC spoofing attacks. A MAC spoofing attack is another link-layer MitM attack in which an adversary sends traffic using another device's MAC address to update the switch's address table, associating the victim's MAC address with the adversary's physical port. Checkpoint prevents this in the same way as for ARP spoofing by allowing devices to *claim* MAC addresses as a byproduct of sending normal network traffic. Once an address has been claimed, no other devices may claim it until the original claim expires. Any packet destined for that address will be blocked from being delivered to any other device than that which claimed the address, thus preventing the MitM attack.

We observe that by binding these low-level network identifiers to each device's true identity, we can *for the first time* provide a simple, robust, and compatible mechanism for eliminating link-layer MitM attacks.

4.2.2 *Dreamcatcher: device permission manager.*

At its core, Dreamcatcher is a firewall. Dreamcatcher blocks all local network traffic by default, allowing only approved connections to proceed. Upon encountering a new network *event*, such as one device sending a packet to another device for the first time, Dreamcatcher will prompt the user to create a *rule* to allow or block the event. Thereafter, for similar events, Dreamcatcher will

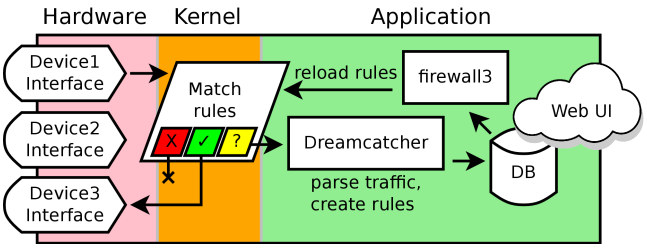


Figure 4: An overview of the Dreamcatcher architecture. Traffic for each device is identified by vNIC. Netfilter makes filtering decisions according to the user's policy. Upon new events, Dreamcatcher creates new rules to affect future network traffic. Users respond to rules through the web UI.

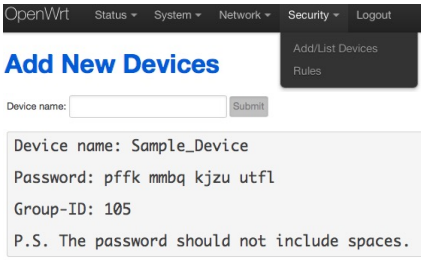


Figure 5: When adding a new device, the web page will populate the RADIUS database with user credentials and display them to the user. The credentials will disappear when the page is reloaded.

follow its existing set of rules without prompting the user. This is a familiar access control model, similar to other demand-driven security applications and recent iterations of mobile permissions systems [3, 13, 43]. Dreamcatcher's architecture is shown in Figure 4.

Users primarily interact with Dreamcatcher's web interface: two additional web pages added to the router's normal configuration interface. New devices are added through the *Add Devices* page, shown in Figure 5. Users are prompted to specify a unique device name, such as "work laptop" or "Alice phone." Dreamcatcher will randomly generate a 16-character alphabetic password for the device and display it to the user, which can be used along with the input device name to connect to the wireless network. Once the user has left the page, the password will never be displayed again. Instead, users are cautioned to never reuse device passwords and to delete and recreate devices in the event of an accidentally forgotten password. We model this password choice after Google's Application-Specific passwords [21] which were designed to be easily entered a single time on devices with reduced accessibility. We note that an alternate password scheme could easily be substituted to meet more or less stringent security requirements, such as extending the password to 20 characters to reach the 64-bits of entropy recommended by NIST's Digital Identity Guidelines [27] for unthrottled attacks.

The Rules page, shown in Figure 6, allows users to view new events and manage the rules they create. New events create pending rules, which are shown at the top of the screen for easy access. Each rule has buttons to declare an *accept* or *reject* verdict, or to *delete* the rule entirely, allowing a future event to regenerate the rule. Approved rules display the previously-chosen verdict, and also allow the user to delete the rule in case the verdict needs to be changed. Rules are described in plain English, not technical terms, and capture relationships between entire devices, not specific types of network traffic.[1] Categorizing network traffic in terms of *intent* is a significant contributor to making the entire system usable in practice.

Types of rules. In attempting to reduce all network traffic into actionable events, we have identified two main types of rules the user must evaluate. The first event type is a *direct connection* between devices, which we present to the user as "<Device A> wants to send messages to <Device B>" using the user-specified device names. This event type is directional. If the rule is accepted, Device A may be able to initiate connections to Device B, but a new connection originating from Device B will be treated as a separate event. This reflects the asymmetric nature of many device interactions, e.g. a user device may need to initiate connections to a lightbulb to change its state, but the lightbulb can be blocked from independently connecting to the user device.

The second event type deals with *service discovery*. Often on local networks, a device will make a general inquiry of the network to see if any other devices support a given service. For instance, your laptop may query the network for any devices that support the IPP (printing) protocol when looking for a network printer. A network printer that receives such a query will then announce its name, the protocols it supports, and its IP address. In Name Poisoning attacks [20], a malicious device can race the legitimate announcement with its own, register the device name with its own IP address, and MitM future connections. Dreamcatcher protects against Name Poisoning attacks by handling announcements as the second event type, presented to the user as, "<Device A> wants to advertise itself on your network as <Advertised Name>." The user can then easily detect and block a malicious device attempting to masquerade as another. We note that devices will typically advertise themselves using a human-readable name, as this name field is often displayed to the user by the application (e.g. "HP Officejet Pro X476"). Currently, Dreamcatcher only supports mDNS-based service discovery, but can be extended to support LLMNR, NetBIOS, SSDP, and other service discovery protocols as well. We have slight variants of these two rule types for broadcast packets and service discovery query packets, respectively.

Companion App. To facilitate real-world use of Dreamcatcher, we created an Android application to deliver real-time rule alerts to the user. This provides immediate feedback, prompting users to make rule decisions immediately after a new event is detected. Dreamcatcher will queue alerts and deliver them to the Android app when it is detected on the local network. The application will listen for an incoming alert, display a notification to the user, and allow the user to quickly Accept or Reject the new rule. We intentionally do not obtain a wake lock in this version. This has the

[1] A single user action often initiates several different types of network connections. In our early prototypes, we found protocol-specific rules to be overly cumbersome.

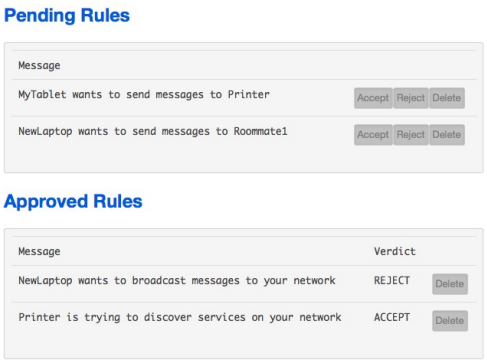


Figure 6: The rules page. Users can control the network policy by accepting and rejecting rules through a plain English UI.

potential downside that a sleeping device will not immediately receive network alerts but avoids draining the device's battery.

Limitations Like any access control system, if the user grants privileges to a compromised device, the device may be able to abuse those privileges to attack other devices on the network. However, Dreamcatcher turns a fully-connected adjacency graph into a sparse graph of network connections, greatly limiting the avenues of attack. Since Dreamcatcher's access control is asymmetric, users can isolate potentially-compromised devices while still allowing their use. Ultimately, for a small usability trade-off, Dreamcatcher provides substantial additional security to devices on the home network, especially to those unable to protect themselves with host-based defenses.

4.2.3 Warden: extending attribution to user-space applications. Warden is a middleware framework that allows high-level applications, such as a web application, to leverage our new attribution mechanism and identify the originating device of any network connection. During the creation of a new application, the developer may specify a *traffic definition*, and Warden will monitor all network traffic conforming to that definition. A traffic definition may consist of a particular network protocol and port, an HTTP URL endpoint, or another identifier of the target network traffic (e.g. all traffic destined for the router's IP address on TCP port 80 or 443). Warden will inform the target application of the originating device's identity for each new connection. This could be done by annotating the packets with the device identity directly, as is done in FlowTags [18]. However, it would be difficult to do so in a uniform way, since different applications may have different requirements and capabilities. For instance, a web application may not be able to access overloaded fields in the Ethernet or TCP headers, and Warden cannot forge HTTP headers within an encrypted HTTPS session. Instead, Warden communicates with the application through a secondary channel, such as a socket or file.

Specifically, we use Warden to solve a usability issue with Dreamcatcher. In the basic Dreamcatcher model, our network is assumed to have an administrator who can make rule decisions as new rules are generated. However, in networks where there is no administrator (e.g. a coffee shop), or the administrator is temporarily unavailable, we wish for non-administrative users to be able to administer their own devices. For instance, if Alice and Bob wish to exchange a file, Alice should be able to allow Bob to connect to her device. Each device should be able to control its own incoming connections, and no device should be able to control incoming connections for other devices. We can do this by filtering the list of rules presented to non-admin devices.

Since the router's web application cannot determine the vNIC associated with each page request, it relies on Warden to identify the device on its behalf. In this case, Warden monitors all traffic destined for the router on TCP ports 80 and 443, serializes the requests, and passes the device IDs to the web application through a shared file. The Rules page, when not accessed by a logged-in admin user, will filter the list of rules to those pertaining to the device accessing the page. Warden thus enables devices to manage their own access control lists in the absence of an administrator.

Name	Model	Role in user study
Laptop1	Macbook Pro	User's main laptop
Laptop2	Macbook Pro (Windows 10)	User's secondary laptop
Laptop3	Dell XPS 13 (Debian)	N/A
Desktop1	Custom (Debian)	N/A
Phone1	Nexus 5x	N/A
Tablet1	Nexus 7	User's tablet
Printer1	Brother DCP-L2540DW	User's printer
Chromecast1	Chromecast	User's media streaming device
Roommate1	Dell Optiplex (Windows 10)	Malicious roommate 1
Roommate2	VMware image (Ubuntu)	Malicious roommate 2
Lightbulb1	Raspberry Pi (Raspbian)	Compromised light bulb bridge

Table 1: Devices used in experimental home network.

5 EVALUATION

We built prototypes of Checkpoint, Dreamcatcher, and Warden on a fork of the OpenWRT Linux framework for embedded devices [12]. Our test platform is a TP-Link WDR4300 wireless router with 8MB of persistent flash storage, 128MB of RAM, and an Atheros AR9344 CPU clocked at 560 Mhz. This router cost under $50 at time of purchase, which we believe makes it a reasonable test platform for an average consumer home. For our testing, we also used a variety of unmodified consumer devices, such as laptops and smartphones, presented in Table 1. We evaluate these security modules on their effectiveness, their usability, and their performance impact.

5.1 Effectiveness

We test a variety of different attacks that may be used on a traditional home or public WiFi network and explore the effectiveness of Checkpoint and Dreamcatcher in defending against these attacks. In each of the following scenarios we demonstrate how a specific attack is thwarted, explore the limits of the defense, and where applicable, identify the conditions under which a similar attack may succeed.

5.1.1 ARP spoofing.
In this scenario, `Laptop3` used the Ettercap [28] attack tool to launch an ARP spoofing attack and intercept packets sent from `Phone1` to `Desktop1`. This attack caused `Laptop3` to send repeated gratuitous ARP replies to `Phone1`, informing it that `Desktop1`'s IP address should be associated with `Laptop3`'s MAC address. Checkpoint, having bound the `Desktop1`'s IP address to `Desktop1`'s identity, categorically blocked `Laptop3`'s ARP replies containing the source

IP address 192.168.1.2. Thus, the ARP spoofing attack was completely blocked with no interruption of network service. We expect all such attacks to be entirely negated by Checkpoint.

We note that, had Checkpoint been missing and the ARP spoofing attack been successful, Dreamcatcher could still have prevented this MitM attack. If the user had not accepted a rule allowing Phone1 to communicate with Laptop3, then despite Phone1's packets being addressed to Laptop3's MAC address, they would have been blocked from delivery and the user would be notified that Phone1 was attempting to communicate with Laptop3. This would downgrade the attack to a Denial of Service.

5.1.2 mDNS spoofing.

In this scenario, Roommate1 used a reimplementation of the mDNS-based MitM attack discovered by Bai et al. [4] to intercept printed documents between Laptop1 and Printer1. When Laptop1 attempted to print, it sent a service discovery packet to the network to find an eligible printer. Roommate1 saw this packet and attempted to race Printer1 with an advertisement for Printer1's device name, "Brother DCP-L2540DW series". If Dreamcatcher had not been running and Roommate1's advertisement arrived first, Laptop1 would have connected instead to Roommate1. However, Dreamcatcher intercepted Roommate1's advertisement packet, blocked it by default, and created a new rule informing the user that Roommate1 was attempting to advertise itself as "Brother DCP-L2540DW series". Thus, the attack was blocked and the user was notified that Roommate1 was attempting to impersonate a printer.

The key observation is that Roommate1 has no business identifying itself to the network as "Brother DCP-L2540DW series". For the most part, we have observed that devices will advertise using a descriptive, human-readable name, as this name is meant to ultimately identify the device to the user. Names chosen by radically different devices, such as a printer and a laptop, should never overlap. However, we can imagine a scenario in which a user has multiple similar devices, such as lightbulbs or same-brand laptops, on the network simultaneously. For example, in a home with both Philips Hue and MiLight smart lightbulbs, one may attempt to impersonate the other. This can be mitigated by the user choosing unique, descriptive names for each device. Ultimately we believe that home networks are generally diverse enough for most mDNS spoofing attacks to be easily detected and mitigated.

5.1.3 Direct attack.

There are many potential direct attacks, ranging from exploits for specific running services to simply logging in to a device via telnet with default credentials. Direct attacks, by our definition, will consist of a malicious device attempting to send some number of packets directly to a victim device to exploit the target vulnerability. These packets, if not already allowed by an existing rule, will trigger a new unicast rule between the malicious and victim devices.

As a stand-in for this category of attacks, we used a Raspberry Pi that we identified to the network as a smart lightbulb. In this toy attack, Lightbulb1 attempted to log in to Laptop1 via SSH with previously-obtained credentials. Dreamcatcher blocked the connection automatically, and because the Lightbulb had no reason to need to connect to Laptop1 in the future, we rejected the rule.

The obvious limitation here is that if Lightbulb1 had already had permission to connect to Laptop1, the malicious request would have been successful. It is easy to focus on hypothetical attacks that could succeed in the presence of Dreamcatcher, but in a real attack scenario, the adversary does not get to choose which devices are allowed to communicate. A user can use Dreamcatcher to limit the possible avenues a compromised device may use to propagate through the network. Our general intuition is that the weak, unpatched devices most likely to be initially compromised will also generally have fewer legitimate reasons to initiate connections to other devices than a multipurpose device like a laptop, and are therefore more likely to be blocked by default. We could also further restrict the attack surface by allowing users to accept rules for short windows of time, thus preventing a one-time setup connection from generating a permanent rule that can be abused in the future. Ultimately, Dreamcatcher *enables* users to block direct attacks, but its efficacy is dependent on the specific network topology and user input. We identify this as an area in need of further research.

5.1.4 Server registration spoofing attack.

In this scenario, Roommate1 launched a MitM attack against the Filedrop application as Laptop1 and Laptop2 attempted to transfer a file. The Filedrop application finds other compatible devices by registering with the Filedrop servers and requesting a list of the IP addresses of any other Filedrop-enabled devices on the local network. We reverse-engineered the Filedrop application and replicated the same device registration and discovery logic in a Node.js script. In this attack, the script periodically retrieved the local device list and upon discovering the target victim, Laptop2, it quickly sent another registration request to the Filedrop server to associate Laptop2's device name with Roommate1's IP address. Without Dreamcatcher, Laptop1 would connect to Roommate1 in place of Laptop2. However, when we performed this test on a Dreamcatcher-enabled network, Laptop1's connection to Roommate1 was blocked by default, generated a new rule, and the Filedrop application failed to connect. Thus, the MitM attack was downgraded to a DoS attack.

Similar to the direct attack scenario, this depends on the existing access control configuration of the network. However, it shows that even when service discovery happens outside the network, Dreamcatcher can still protect against otherwise-invisible attacks.

5.2 Usability

While Checkpoint's defense is completely transparent to the user, the effectiveness of Dreamcatcher is directly dependent on how it is used. For instance, it is critical that users do not share credentials between devices, so our device enrollment process encourages good behavior by making new device enrollment easier than retrieving existing credentials. Empirically determining the real-world usability of Dreamcatcher is difficult, since it has not yet been distributed to a wide audience that could provide direct feedback. In light of this, we have performed two different and complimentary usability studies to test the overall usability of the system. Both of these studies were conducted after obtaining waivers from our University's Institutional Review Board.

The first, a survey with participants recruited through Amazon's Mechanical Turk [1], primarily tested whether users from a wide variety of different computer and networking experience levels could understand the security and privacy implications of Dreamcatcher and respond to alerts properly. We tested both whether users were

Experience Level		Network			
		1: Limited	2: Some	3: Strong	Total
Computer	1: Beginner	0	0	0	0
	2: Average	4	27	0	31
	3: Power	1	38	6	45
	4: Beginner Dev	0	8	10	18
	5: College Degree	0	2	11	13
	Total	5	75	27	107

Table 2: Number of Mechanical Turk survey participants, grouped by self-reported computer and network experience levels.

Scenario	Category	Normal Users	Rushed Users	Abnormal Users
1	Setup	68/95 (72%)	2/9 (22%)	0/3 (0%)
2	Setup	82/95 (86%)	2/9 (22%)	0/3 (0%)
3	Normal Use	90/95 (95%)	3/9 (33%)	1/3 (33%)
4	Attack	63/95 (66%)	3/9 (33%)	2/3 (67%)
5.1	Normal Use	87/95 (92%)	7/9 (78%)	1/3 (33%)
5.2	Normal Use	74/95 (78%)	6/9 (67%)	3/3 (100%)
5.3	Normal Use	94/95 (99%)	8/9 (89%)	3/3 (100%)
6	Attack	78/95 (82%)	6/9 (67%)	3/3 (100%)
7	Attack	86/95 (91%)	2/9 (22%)	2/3 (67%)
Mean Time (mm:ss) [Std. Dev]		24:36 [14:24]**	6:48 [1:44]	27:04 [26:26]

Table 3: Proportion of scenarios completed successfully by Mechanical Turk workers. **We exclude the survey time from seven outliers of our normal user population with survey times in excess of 80 minutes.

Experience Level		Network		
		1: Limited	2: Some	3: Strong
Computer	1: Beginner	69.47%	71.11%	86.96%
	2: Average	69.47%	71.11%	86.96%
	3: Power	70.15%	71.21%	86.96%
	4: Beginner Dev	81.48%	81.48%	84.21%
	5: College Degree	81.82%	81.82%	88.89%

Table 4: Percentage of normal users with at least the listed experience levels who correctly answered *all* normal use survey questions.

able to block various attacks on the network as well as accept the requisite rules to enable normal network functionality. The second, a small in-person user-study using our experimental environment, tested whether users could effectively use Dreamcatcher in an end-to-end setting. That is, in addition to making appropriate decisions about which networking rules to accept or reject, could the user remember all of the steps to properly add a new device to the network, make the connection between a blocked-by-default user-initiated action and corresponding new rules, and all of the other myriad intermediate steps required to actually use Dreamcatcher in the real world?

We also note anecdotally that our lead author has been using this system in their home network for several months. Since the initial setup, they have not needed to manage a single rule and thus our experience has shown that this system is not cumbersome to use in the real world.

5.3 Mechanical Turk survey

Amazon's Mechanical Turk is an online labor market in which Requestors can post tasks to be completed, and Workers can complete tasks for payment. We used Mechanical Turk to recruit participants for our survey, restricting participants to those with over 500 completed tasks and a 95% acceptance rate but with no diversity restrictions. In Mechanical Turk, workers are typically paid a flat fee for each task they complete. Therefore, workers are incentivized to complete tasks as quickly as possible to maximize their hourly income. Since we wished for our participants to respond thoughtfully to our survey, we structured payment in terms of a base rate to be paid for completion of the survey, with a bonus to be paid for "correct" responses to the survey questions. We manually reviewed and graded all survey responses. Of 108 workers who accepted our task, 95 completed the survey normally, 9 we designated as *rushed* due to completion times under nine minutes[2], 3 gave abnormal answers leading us to believe they didn't fully understand the survey itself, and 1 was eliminated for failing to complete the survey in good faith.

At a high level, our survey was constructed of the following components: (1) A brief overview of Dreamcatcher.(2) A Q&A section to record the participant's self-reported computer and network experience levels, the results of which are shown in Table 2. (3) A walkthrough of the Dreamcatcher user interface and description of how to add devices to the network and make rules decisions. (4) A short training module to reinforce the concepts of adding devices to the network, accepting necessary rules, and blocking potentially-malicious network actions. None of the training modules reflected a subsequent storyline scenario. (5) A series of scenarios in which we presented a fictional storyline and asked participants to respond to the network events that would have occurred, the results of which are listed in Table 3. (6) A brief post-survey exit questionnaire.

Scenarios 1 and 2 were free-response questions about the steps necessary to add devices to the network, one before and one after the training module, although both were after a description of how to add devices to the network. They were graded subjectively based on whether we believed the user would be able to properly add new devices to the network. Correct and incorrect answers were clearly differentiable in the vast majority of cases. An answer of, "go in phone settings and connect to my wifi connection" was marked incorrect whereas an answer of, "Using my laptop I need to log into the router and add another device for my smartphone" was marked correct. Scenarios 3 and 5 presented the user with scenarios in which they needed to transfer a file and print a document, respectively, and asked which, if any, rules they would need to accept, reject, or delete to accomplish the stated objective. Responses were graded correct if the participant accepted all of the requisite rules to enable the network function to succeed. Scenarios 4, 6, and 7 presented the user with scenarios in which the previously-working file transfer, a subsequent print job, and a YouTube video failed due to malicious interference (Section 5.1) and again asked which, if any, rules would need to be adjusted. Responses were graded correct if the participant did *not* accept any rules that would allow the malicious behavior to succeed. We intentionally did not point out any malicious behavior, but only that the services in question were malfunctioning.

From our normal user survey results, it is clear that the majority of users were able to successfully navigate most of the scenarios

[2]There was a natural lull of completion times around the 9-minute mark, and we noticed a significant difference in the quality of answers above and below this mark.

presented to them. Most importantly, almost all users were able to properly identify the rules they needed to accept in order to successfully transfer a file and print a document.Only 14% of users who correctly enabled device communication also granted extra privileges to other benign devices on the home network.

As shown in Table 4, 69% of all users were able to answer *all* of the normal use questions correctly, with higher percentages among more technical participants. With very little training, and no way to learn from their mistakes as would be possible in the real world, the majority of our participants has no difficulty using Dreamcatcher to accept rules for everyday activities. This is significant, as any user who can succeed in normal device operations has very little disincentive to using Dreamcatcher and is able to benefit from the ability to resist attacks, even if some attacks can still succeed for some users.

5.4 In-person user study

To address the open questions from our Mechanical Turk study, we also performed a small in-person user study using our experimental setup. We primarily wish to evaluate whether users can operate Dreamcatcher properly in a real-world setting, including the various intermediate steps necessary to maintain a functioning network. We recruited 11 participants from the undergraduate and graduate students from our university's computer science department. None of these participants had previously been briefed on Dreamcatcher and had varying levels of networking experience, although all had significant computer experience. Each participant was made the admin of our experimental network and, after a brief tutorial on the system, asked to perform several everyday tasks.

Prior to running the user study with each participant, we reset the experimental network to its default state. In this state, the devices `Chromecast1`, `Laptop2`, `Printer1`, `Roommate1`, `Roommate2`, and `Lightbulb1` were already connected to the network, with default rules rejecting any communication between these devices. Our Android application was pre-installed and configured on `Tablet1`. Communication between these devices was not evaluated as part of the user study, and creating rules to block communication between them is representative of an existing Dreamcatcher network that has reached a steady state with regard to rule creation. We did not inform participants that we would be launching attacks on the network.

During the user study, participants were asked to perform the following *actions*: 1) Use `Laptop2`'s web browser to access Dreamcatcher's web interface and add `Tablet1` to the network. 2) Use `Tablet1` to cast a Youtube video to `Chromecast1`. 3) Add `Laptop1` to the network. 4) Transfer a file from `Laptop2` to `Laptop1` using the Filedrop application three times. 5) Use `Laptop1` to perform a Google search (which would complete successfully). 6) Use `Laptop1` to send a file to `Printer1` for printing three times. 7) Use `Laptop1` to perform another Google search (which would be affected by the ARP spoofing attack). During actions 1 and 2, we assisted the user in adding a new device to the network and managing the new rules that took effect as `Tablet1` attempted to cast a video to `Chromecast1`. We consider these steps a tutorial in how to use Dreamcatcher and do not include them in our user study results, shown in Table 5. Afterwards, the user was responsible for performing actions 3-7 without further guidance.Meanwhile, hidden from the user, we launched the four network attacks described in Section 5.1 to see whether they would inadvertently allow any of the attacks to succeed. We asked participants to repeat actions 4 and 6 three times each so that both correct function and an attack against the service could be attempted.

Point category	Action	User #				
		1,5-10	2	3	4	11
Device Setup	Add Laptop1 using the Dreamcatcher web interface	✓	✓	✓	✓	✓
Normal behavior support	Accept the rule for transferring file from Laptop1 to Laptop2 using FileDrop	✓	✓	✓	✓	✗
	Accept the rule for Laptop1 to send mDNS discovery message	✓	✗	N/A	✓	✓
	Accept the rule for Printer1 to send mDNS advertisement message	✓	✓	N/A	✓	✓
	Accept the rule for Laptop1 to connect to Printer1 for printing	✓	✓	N/A	✓	✓
Defense	Reject/ignore rules for malicious login attempt from Lightbulb1	✓	✓	✓	✓	✓
	Reject/ignore rules for FileDrop MitM attack from Roommate1	✓	✓	✓	✓	✓
	Reject/ignore rules for Printer MitM attack from Roommate1	✓	✓	✓	✓	✓
	Reject/ignore rules for ARP spoofing attack from Roommate2	✓	✓	✓	N/A	✓

Table 5: In-person user study results. "N/A" indicates that an external factor interfered with the test case.

All users were able to add Laptop1 to the network without difficulty. In the four rules necessary for normal behavior support, only users 2 and 11 had difficulty performing their everyday actions. These are marked as failures in Table 5. In user #2's case, `Laptop1` attempted to discover services on the network prior to the user reaching action 6, printing a file, and they overzealously blocked what they thought was an unnecessary rule at that earlier time. When user #2 reached the point in the study where they had to print, we had to remind them to revisit the list of approved rules, after which they were able to identify the blocked rule, accept it, and print. In user #11's case, when attempting to transfer a file using the Filedrop application and Dreamcatcher blocked the initial connection, they forgot that they would need to review the rule list to allow connections between the two laptops. `Tablet1` did make an audible beep and display a notification that a new rule had taken effect, but the user missed this cue. After a quick reminder to check the rule list, they were able to transfer a file, and had no difficulties with the rest of the study. We note that all test cases marked successful in Table 5 were completed without help.

In no test case did a user allow an attack to succeed. Dreamcatcher blocks new connections by default, so if the user ignores a rule they do not understand the attack will still be blocked. Since this is by design, we do not differentiate between cases where a user ignored or explicitly rejected the rules generated by an attack.

In two instances, external factors interfered with a test case. In user #3's case, `Laptop1` was unable to print. Despite our intervention in which we reset the router state to a permissive rule set, `Laptop1` could not connect to `Printer1`. We asked user #3 to continue on with the next action and we omit that test case from our user study results. In user #4's case, the ettercap attack tool running on `Roommate2` did not send the requisite ARP packets to

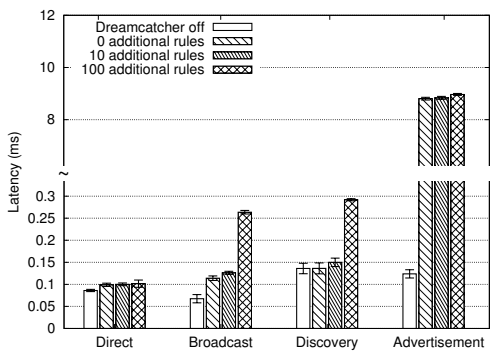


Figure 7: First-packet latency. Service discovery often takes several seconds and the ~9 ms of additional latency for advertisement packets does not degrade the user experience.

the network when run. Thus, the user did not experience the ARP spoofing attack and thus this test case is omitted. In both cases, after power cycling the affected device and resetting the network to its initial test state, later tests did not exhibit the same issues.

Very few users had any difficulty in identifying and accepting the rules necessary for everyday tasks to succeed, and those who did have difficulty were able to solve their issues with minimal reminders. This suggests that both the user interface and immediate feedback from push-notifications are conducive to effective use of a permissions-based model, and perhaps even the few issues we saw could be addressed with additional user training. None of the users allowed an attack to succeed, indicating that Dreamcatcher is effective in protecting the local network.

5.5 User Study Limitations

We would like to make it clear that we draw no statistical conclusions from our user studies. In particular, our in-person user study is both biased towards technical users and not sampled from a random population. As is clear from Table 2, more technical users will tend to exhibit larger benefits from using our system. Similarly, the Mechanical Turk population has been shown to be more technologically-literate than the general population. [6] However, this technical bias is also likely to be reflected among users who ultimately decide to adopt this system. Since our goal is to *enable* users to better defend their networks, our user studies help demonstrate that this is feasible, at least among some portion of the population.

5.6 Performance

As Dreamcatcher and Checkpoint affect every packet that traverses the router, it is important that it not affect the user experience by introducing performance overhead. The two aspects of performance we measure are first-packet latency and overall throughput. First-packet latency is important because it determines the responsiveness of services. Throughput is a measurement of how long it takes the user to perform data-intensive tasks, such as transferring a file. Since the filtering for both Checkpoint and Dreamcatcher is done through the *netfilter* framework entirely in the kernel, we expect the performance overhead to be minimal. Due to Checkpoint and Dreamcatcher sharing a common filtering platform and

System configuration	TCP traffic throughput (Mbps)	
	Average	Standard deviation
Dreamcatcher off	51.80	2.44
0 additional rules	52.99	2.08
10 additional rules	52.92	2.39
100 additional rules	51.60	2.70

Table 6: TCP traffic throughput measurement over 20 iterations with and without Dreamcatcher running.

Dreamcatcher's rules being substantially more complex than those of Checkpoint, we have taken a worst-case approach and present our results using Dreamcatcher rules.

5.6.1 First-packet Latency.

To measure first-packet latency, we sent various requests between a desktop and laptop, both with Dreamcatcher completely disabled and enabled with various numbers of rules in effect. Using *tcpdump* on the router, we captured timestamps for all packets both as they entered through the desktop's network interface and left through the laptop's network interface. Thus, we can rule out any delays from wireless interference or retransmission. We test Dreamcatcher under four configurations: with Dreamcatcher completely disabled, Dreamcatcher enabled but with no additional rules, with 10 additional rules, and with 100 additional rules. We evaluate first-packet latency differently for each category of network traffic, as in some cases the traffic is handled differently by Dreamcatcher. In all cases with Dreamcatcher enabled, a rule exists to *accept* the specific network traffic we evaluate, since otherwise the traffic would be blocked. Our results over 100 iterations are detailed in Figure 7.

For **direct** packets, we send ICMP echo (ping) requests from the desktop to laptop.[3] With Dreamcatcher disabled, these packets are immediately sent to the laptop. With Dreamcatcher enabled, these packets traverse the netfilter chain for direct packets and match the allowed rule. As we add additional direct rules prior to the *accept* rule, the overhead increases by only 0.016 milliseconds on average between the 'Dreamcatcher off' and '100 rule' tests.

For **broadcast** packets, we similarly send ICMP echo requests from the desktop, and measure them from the laptop, but these packets are sent to the broadcast address and to every other device on the network. As the packets hit the routing layer, they are split into multiple packets – one for each device on the network – which we believe accounts for the slightly increased latency for broadcast packets. Still, the average overhead introduced between the 'Dreamcatcher off' and '100 rule' tests is only 0.196 milliseconds.

For **discovery** packets, we replay an mDNS discovery packet using the *tcpreplay* [41] utility. With Dreamcatcher disabled, this packet is immediately accepted. With Dreamcatcher enabled, we see very similar latency increases to the broadcast packet test. This is to be expected, since the discovery packet it sent to a multicast IP address and similarly split into multiple packets, one for each device on the network. The average overhead introduced between the 'Dreamcatcher off' and '100 rule' tests is 0.156 milliseconds.

For **advertisement** packets, we also use tcpreplay to replay an mDNS advertisement packet. We note that the mDNS advertisement packet we used contained four answer fields and two unique device names, requiring our mDNS kernel module to match the

[3] ICMP echo requests do not match the ESTABLISHED or RELATED netfilter states, so will not short-circuit the rule list [2].

packet against multiple approved device names. This is representative of real-world advertisement packets and not a best-case scenario. We notice 8.682 milliseconds of additional latency as soon as Dreamcatcher is enabled, with only a 0.161 millisecond increase as additional rules are added. We believe this dramatic increase is due to the complexity of parsing the mDNS name structure, which has variable length and can contain back-references at any point. We ensure that for each device, the mDNS kernel module will only need to parse the name structure once by bundling all approved names into a single netfilter rule and ensuring that that rule is placed before any rules for rejected names. If the packet is not accepted, it will either be rejected or sent to Dreamcatcher, in which case any additional latency is irrelevant since the packet will be blocked. Thus, the 100 additional rules we add prior to the target *accept* rule must be for different devices and can be quickly passed without activating the mDNS kernel module. We also note that for service discovery 9 milliseconds is unnoticeable in practice. The mDNS specification, RFC 6762 [10], repeatedly mentions that mDNS responders should delay their responses by up to 500 milliseconds, and in practice, service discovery often takes several seconds. Thus, we consider Dreamcatcher's first-packet latency well within acceptable bounds.

5.6.2 *Throughput.*

To measure throughput, we used the *iperf* [15] utility, a standard networking bandwidth measurement tool. We installed iperf on both our desktop and laptop, and performed twenty 10-second TCP bandwidth measurements for each of the four Dreamcatcher configurations. The results are shown in Table 6.

The first packet of any new connection must traverse the netfilter rule list, which, as shown in our first-packet latency results above, introduces negligible latency. Thereafter, additional packets in the connection will recognized as part of an existing connection, short-circuit the rule list, and be accepted immediately. Despite the lack of connection for UDP and ICMP flows, iptables will maintain state for recent UDP and ICMP packets and subsequent packets will short-circuit the rule list as well [2]. Once a connection is allowed, we expect Dreamcatcher to not affect the router's throughput at all. This hypothesis is supported by our test results, which show negligible change in bandwidth across all four test cases.

6 CONCLUSION

Adding attribution to commodity small networks makes them demonstrably safer. In this work, we introduce a new mechanism for strong attribution on local networks, and develop three prototype security modules that can protect devices against several categories of attacks. We evaluate these security modules, and show that they block ARP spoofing, name poisoning, and in many cases even direct attacks on vulnerable devices, while maintaining compatibility with existing network protocols and normal network functions. Through two user studies, we demonstrate that the majority of surveyed users can operate these security modules properly with minimal training, and block almost all attacks, even when unaware. Our solutions introduce only a fraction of a millisecond of additional first-packet latency for most new connections, and negligible throughput overhead.

REFERENCES

[1] Amazon. 2017. Mechanical Turk is a marketplace for work. (2017). https://goo.gl/U8xSuL
[2] Oskar Andreasson. 2008. Iptables. (2008). https://goo.gl/ADcLts
[3] Apple. 2017. Requesting Permission. (2017). https://goo.gl/gFrMtT
[4] Xiaolong Bai et al. 2016. Staying Secure and Unprepared: Understanding and Mitigating the Security Risks of Apple ZeroConf. In *IEEE Symposium on Security and Privacy*.
[5] Hanan Be'er. 2016. *Metaphor: A (real) real-life Stagefright exploit.* Technical Report. https://goo.gl/NFTb3U
[6] Tara S. Behrend, David J. Sharek, Adam W. Meade, and Eric N. Wiebe. 2011. The viability of crowdsourcing for survey research. *Behavior Research Methods* (2011). https://goo.gl/zoanxU
[7] Nick Bilton. 2016. Nest Thermostat Glitch Leaves Users in the Cold. (2016). https://goo.gl/tGeWjU
[8] Bill Brenner. 2017. WannaCry: the ransomware work that didn't arrive on a phishing hook. (2017). https://goo.gl/SBWyXH
[9] D. Bruschi, A. Ornaghi, and E. Rosti. 2003. S-ARP: a Secure Address Resolution Protocol. In *Proceedings of the 19th Annual Computer Security Applications Conference*. ACSAC. https://goo.gl/Kvyn4G
[10] S. Cheshire and M. Krochmal. 2013. *Multicast DNS.* RFC 6762. https://goo.gl/nVPq2G
[11] Soteris Demetriou et al. 2017. HanGuard: SDN-driven protection of smart home WiFi devices from malicious mobile apps. In *10th ACM Conference on Security and Privacy in Wireless and Mobile Networks*. https://goo.gl/npPaKH
[12] The OpenWRT developer team. 2017. OpenWRT: Wireless Freedom. (2017). https://openwrt.org/
[13] Objective Development. 2017. Little Snitch 3. (2017). https://goo.gl/fdjyrj
[14] Paul Duckin. 2016. DirtyCOW Linux hole works on Android too - "root at will". (2016). https://goo.gl/rzAhpx
[15] Jon Dugan, Seth Elliott, Bruce A. Mah, Jeff Poskanzer, and Kaustubh. 2017. iPerf - The network bandwidth measurement tool. (2017). https://iperf.fr/
[16] Tim Enwall and Mike Soucie. 2016. A letter from Revolv's founders. (2016). https://goo.gl/EGsLaf
[17] F-Secure. 2017. F-Secure Sense. (2017). https://goo.gl/TpGjcF
[18] Seyed Kaveh Fayazbakhsh, Luis Chiang, Vyas Sekar, Minlan Yu, and Jeffrey C. Mogul. 2014. Enforcing Network-Wide Policies in the Presence of Dynamic Middlebox Actions using FlowTags. In *11th USENIX Symposium on Networked Systems Design and Implementation*. https://goo.gl/Ha1gYG
[19] Erica Fink and Laurie Segall. 2013. Your TV might be watching you. (2013). https://goo.gl/Wkdt5P
[20] GnuCitizen. 2008. Name (mdns) poisoning attacks inside the lan. (2008). https://goo.gl/tvB7nB
[21] Google. 2017. Sign in using App Passwords. (2017). https://goo.gl/JxoseS
[22] Gao Jinhua and Xia Kejian. 2013. ARP spoofing detection algorithm using ICMP protocol. In *International Conference on Computer Communication and Informatics*. https://goo.gl/6HVxYM
[23] Stephen T. Kent. 1993. Internet Privacy Enhanced Mail. *Commun. ACM* (1993). https://goo.gl/vRjxQ4
[24] Tiffany Hyun-Jin Kim et al. 2014. Lightweight source authentication and path validation. In *ACM SIGCOMM Computer Communication Review*. https://goo.gl/k2kHMq
[25] Jad Naous et al. 2011. Verifying and enforcing network paths with icing. In *Proceedings of the Seventh COnference on emerging Networking EXperiments and Technologies*. https://goo.gl/RjVc2Y
[26] United States Department of Homeland Security. 2017. Best Practices For Using Public Wi-Fi Tip Card. Brochure. (2017). https://goo.gl/QxuANv
[27] National Institute of Standards and Technology. 2017. Authenticator and Verifier Requirements. (2017). https://goo.gl/RQbiUr
[28] Alberto Ornaghi, Marco Valleri, Emilio Escobar, and Eric Milam. 2001. Ettercap. (2001). https://goo.gl/X6vzn8
[29] Andre Ortega, Xavier Marcos, Luis Chiang, and Cristina Abad. 2009. Preventing ARP Cache Poisoning Attacks: A Proof of Concept using OpenWrt. In *Latin American Network Operations and Management Symposium*. https://goo.gl/6cufhN
[30] Vern Paxson. 1998. Bro: A System for Detecting Network Intruders in Real-Time. In *USENIX Security Symposium*.
[31] Vivek Ramachandran and Sukumar Nandi. 2005. Detecting ARP Spoofing: An Active Technique. In *Information Systems Security*. ICISS. https://goo.gl/eCf4xo
[32] Symantec Security Response. 2016. Mirai: what you need to know about the botnet behind recent major DDoS attacks. (2016). https://goo.gl/Dm66by
[33] O. M. Ritchie and K. Thompson. 1978. The UNIX time-sharing system. In *The Bell System Technical Journal*. https://goo.gl/hU2SPr
[34] Martin Roesch et al. 1999. Snort: Lightweight Intrusion Detection for Networks. In *LISA*.
[35] Franziska Roesner. 2017. Designing Application Permission Models that Meet User Expectations. *IEEE Security Privacy* (2017). https://goo.gl/JfMM1c

[36] Eyal Ronen, Colin O'Flynn, Adi Shamir, and Achi-Or Weingarten. 2016. IoT Goes Nuclear: Creating a ZigBee Chain Reaction. (2016). https://goo.gl/benz2C
[37] J. H. Saltzer, D. P. Reed, and D. D. Clark. 1984. End-to-end Arguments in System Design. *ACM Trans. Comput. Syst.* (1984). https://goo.gl/xBMYFZ
[38] Jennifer G. Steiner, Clifford Neuman, and Jeffrey I. Schiller. 1988. Kerberos: An Authentication Service for Open Network Systems. (1988). https://goo.gl/Rj8GQm
[39] Jing Tian, Kevin R. B. Butler, Patrick D. McDaniel, and Padma Krishnaswamy. 2015. Securing ARP From the Ground Up. In *Proceedings of the 5th ACM Conference on Data and Application Security and Privacy*. https://goo.gl/goRWWP
[40] M. V. Tripunitara and P. Dutta. 1999. A middleware approach to asynchronous and backward compatible detection and prevention of ARP cache poisoning. In *15th Annual Computer Security Applications Conference (ACSAC '99)*. IEEE. https://goo.gl/3fWLyX
[41] Matt Undy, Matt Bing, Aaron Turner, and Fred Klassen. 2013. Tcpreplay: Pcap editing & replay tools for *NIX. (2013). https://goo.gl/8AB5hq
[42] Victor van der Veen et al. 2016. Drammer: Deterministic Rowhammer Attacks on Mobile Platforms. In *23rd ACM Conference on Computer and Communication Security (CCS '16)*. https://goo.gl/Y4wwMr
[43] Primal Wijesekera et al. 2015. Android Permissions Remystified: A Field Study on Contextual Integrity. In *24th USENIX Security Symposium*. https://goo.gl/DhMfeB
[44] WikiLeaks. 2017. Vault 7: CIA Hacking Tools Revealed. (2017). https://goo.gl/1a1yS8
[45] Zhi-Kai Zhang et al. 2014. IoT Security: Ongoing Challenges and Research Opportunities. In *IEEE 7th International Conference on Service-Oriented Computing and Applications*. IEEE. https://goo.gl/DLntPG