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 intra-network attacks, such as ARP spoofing. With the increased prevalence of oft-compromised Internet-of-Things (IoT) devices now taking up residence in homes and small businesses, the potential for abuse has never been higher. In this work, we present a novel mechanism for strongly attributing local network traffic to its originating principal, fully-compatible with existing legacy devices. 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. 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 devices and therefore failed to be adopted. Our prototype imposes negligible performance overhead, runs on an inexpensive commodity router, and retains full compatibility with modern and legacy devices.

CCS CONCEPTS

• **Security and privacy** → **Access control**; *Firewalls*; • **Networks** → *Wireless access points, base stations and infrastructure*;

KEYWORDS

Dreamcatcher; Checkpoint; WPA; vNIC; ARP Spoofing; Name Poisoning

ACM Reference format:
Jeremy Erickson, Qi Alfred Chen, Xiaochen Yu, Erinjen Lin, Robert Levy, and Z. Morley Mao. 2018. No One In The Middle: Enabling Network Access Control Via Transparent Attribution. In *Proceedings of 2018 ACM Asia Conference on Computer and Communications Security, Incheon, Republic of Korea, June 4–8, 2018 (ASIA CCS '18),* 8 pages.
https://doi.org/10.1145/3196494.3196498

ACM ISBN 978-1-4503-5576-6/18/06…$15.00
https://doi.org/10.1145/3196494.3196498

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 local-device permission systems [18], 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 [21], ticket-granting [22], and the Public Key Infrastructure [13]. 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. For instance, in the ARP spoofing attack, an attacker associates its MAC address with a victim's IP address to intercept the victim's packets, 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 robust intrusion detection. 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 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 [10, 17, 20, 26]. The Vault 7 leaks [25] 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 as 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 [3]. 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 [14]. 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.

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 differentiate flows between devices, even when adversaries spoof their identifiers. 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 resides below the link layer of the network stack and thus requires no changes to existing devices or protocols. It 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 present two new security modules: *Checkpoint*, which eliminates Man-in-the-Middle (MitM) attacks at the link layer, and *Dreamcatcher*, which enforces an on-demand access control policy for devices on the network.

Users primarily interact with Dreamcatcher, a *user-driven* access control system [19], 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 creates 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 is 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 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.
- A prototype implementation of our attribution mechanism and new security modules which runs on a $50 commodity router and is released as open source.

2 THREAT MODEL

In this work, we consider any internal 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 to the router or other devices on the network.

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.

3 METHODOLOGY

3.1 Attribution mechanism

The MAC and IP addresses used in all low-level protocols such as the 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 identifiers with more secure alternatives [2, 4, 6] 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.

3.1.1 *Assigning unique device credentials.*

In most small networks using **WPA Personal** network authentication, credentials consist of the SSID (network name) and a pre-shared key (PSK) 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 PSK represents a unique credential for a *single device*, so long as that credential is not shared. Modern consumer routers often support the ability to run multiple wireless networks using the same physical radio, each with a unique PSK. We can assign unique credentials to each device by creating them separate wireless networks

Enterprise environments must often support hundreds or thousands of clients simultaneously and so use one of several enterprise protocols. All **WPA Enterprise** protocols assign each user their own credentials, typically in the form of a username and password, which we can use directly.

These two credential schemes 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 by all devices, particularly *legacy* devices such as network printers and streaming media players. 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 15 available wireless networks from two radios to support a single legacy device each. We believe this should be sufficient for the majority of small network use cases.

3.1.2 *Binding device credentials to vNICs.*

After assigning a unique credential to each device, we must provide a way for applications to attribute flows to devices. For both of the credentialing techniques presented above, we split each device's traffic out into a separate vNIC within the router based on the credentials provided. For WPA Personal networks, each generates its own network interface: `wlan0`, `wlan1`, etc. These 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. We can create a new VLAN with associated vNIC for each device. 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 when using VLAN isolation. 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 clients enter and leave the network. 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 1.

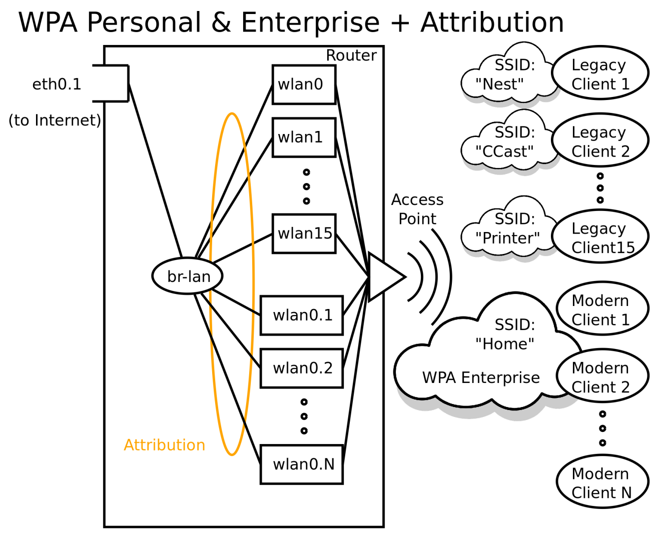

Figure 1: By combining both techniques for attribution on WPA Personal and WPA Enterprise networks, we can support an effectively unlimited number of modern devices and 15 legacy devices.

3.2 Security applications

Leveraging this support for strong attribution, we are able to design and implement security modules to defend against local network attacks at higher layers of the network stack. We present **Checkpoint**, a completely transparent solution to ARP and MAC spoofing attacks, and **Dreamcatcher**, an access control system for the commodity small network.

3.2.1 *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. In ARP spoofing attacks, a adversary associates the target IP address with its own MAC address so that packets are rerouted to itself at the link layer.

In traditional networks, this is a difficult issue to address since MAC addresses can be spoofed. 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 via a similar mechanism.

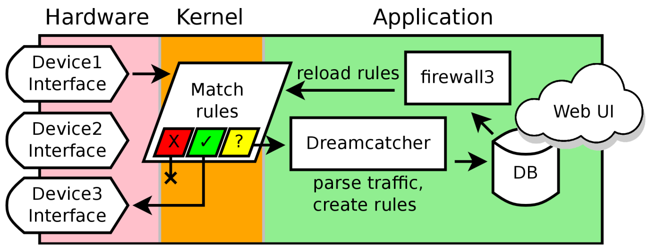

Figure 2: Dreamcatcher architecture. Dreamcatcher creates new rules on demand. Users change policy with the web UI.

By binding these low-level network identifiers to each device's true identity, we can provide a simple, robust, and compatible mechanism to *eliminate* link-layer MitM attacks.

3.2.2 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 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 [1, 8, 24]. Dreamcatcher's architecture is shown in Figure 2.

Users 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. 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 [12] 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 [15] for unthrottled attacks.

The Rules page 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 *accept reject*, or *delete* 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. Details of this categorization can be found in Appendix A.

[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.

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.

4 EVALUATION

We built prototypes of Checkpoint and Dreamcatcher on a fork of the OpenWRT Linux framework for embedded devices [7]. 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, listed in Appendix B. We evaluate these security modules on their effectiveness, their usability, and their performance impact.

4.1 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.

4.1.1 ARP spoofing.
In this scenario, `Laptop3` used the Ettercap [16] tool to launch an ARP spoofing attack and MitM the connection between `Phone1` and `Desktop1`. `Laptop3` sent repeated gratuitous ARP replies to `Phone1`, informing it that `Desktop1`'s IP address should be associated with `Laptop3`'s MAC address. Checkpoint, having already bound the `Desktop1`'s IP address to `Desktop1`'s identity, blocked `Laptop3`'s ARP replies containing `Desktop1`'s IP address in the source address field. Thus, the ARP spoofing attack was completely blocked.

4.1.2 mDNS spoofing.
In this scenario, `Roommate1` used a reimplementation of the mDNS-based MitM attack discovered by Bai et al. [2] 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.

4.1.3 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. `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.

4.1.4 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.

4.2 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. To evaluate the usability of Dreamcatcher, we turn to a study using Mechanical Turk as a stand-in for real users. This study was conducted after obtaining a waiver from our University's Institutional Review Board.

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, 3 gave abnormal answers indicating 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 composed of the following components: (1) A brief overview of Dreamcatcher. (2) A form for the participant's self-reported computer and network experience levels, included in Appendix C. (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 introduce the concepts of adding devices to the network, accepting necessary rules, and blocking potentially-malicious network actions. None of the training scenarios 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, shown in Table 1. (6) A brief post-survey exit questionnaire. Details of these scenarios and how they were graded are included in Appendix D.

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 1: 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.

From our normal user survey results, it is clear that the majority of users were able to successfully navigate most of the scenarios 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. 69% of all users were able to answer *all* of the normal use questions correctly, and among more technical participants the success rate reached 88%. 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 had 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.

4.3 Performance

As Dreamcatcher and Checkpoint affect every packet that traverses the router, it is important that it not introduce overhead that affects the user experience. We measure both first-packet latency and overall throughput. 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 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.

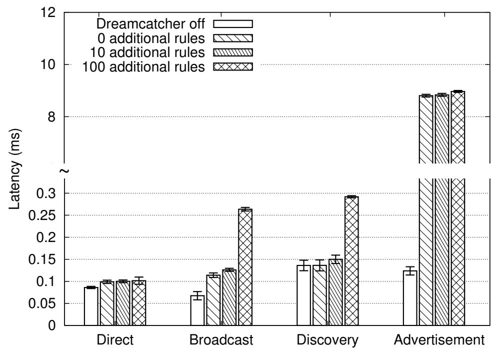

Figure 3: First-packet latency. Service discovery often takes several seconds and the ~9 ms of additional latency for advertisement packets does not degrade the user experience.

4.3.1 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 overhead from wireless interference or retransmission. We tested 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 3. Details of each experiment can be found in Appendix E.

We note that for service discovery 9 milliseconds is unnoticeable in practice. The mDNS specification, RFC 6762 [5], repeatedly mentions that mDNS responders should delay their responses by up to 500 milliseconds, and in practice, service discovery often takes several seconds. Dreamcatcher's first-packet latency is unnoticeable.

4.3.2 Throughput.
To measure throughput, we used the *iperf* [9] utility. We installed iperf on both our desktop and laptop, and performed twenty 10-second TCP bandwidth measurements for each of the four Dreamcatcher configurations. We observed average bandwidths of 51.80 mbps, 52.99 mbps, 52.92 mbps, and 51.60 mbps for the Dreamcatcher off and 0/10/100 additional rules tests, respectively, with standard deviations ranging from 2.08 to 2.70 mbps. In other words, no significant decrease in bandwidth with Dreamcatcher enabled.

5 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 two 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. We demonstrate that the majority of surveyed users can operate these security modules properly with minimal training, and block most 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.

6 ACKNOWLEDGEMENTS

The authors thank George Theodorakopoulos for his shepherding of this paper and the anonymous reviewers for their valuable feedback. This work was in part supported by NSF under the awards CNS-1318306 and CNS-1526455, as well as ONR under the awards N00014-14-1-0440 and N00014-18-1-2020.

REFERENCES

[1] Apple. 2017. Requesting Permission. (2017). https://goo.gl/gFrMtT
[2] 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.*
[3] Bill Brenner. 2017. WannaCry: the ransomware work that didn't arrive on a phishing hook. (2017). https://goo.gl/SBWyXH
[4] 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
[5] S. Cheshire and M. Krochmal. 2013. *Multicast DNS.* RFC 6762. https://goo.gl/nVPq2G
[6] 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
[7] The OpenWRT developer team. 2017. OpenWRT: Wireless Freedom. (2017). https://openwrt.org/
[8] Objective Development. 2017. Little Snitch 3. (2017). https://goo.gl/fdjyrj
[9] Jon Dugan, Seth Elliott, Bruce A. Mah, Jeff Poskanzer, and Kaustubh. 2017. iPerf - The network bandwidth measurement tool. (2017). https://iperf.fr/
[10] Erica Fink and Laurie Segall. 2013. Your TV might be watching you. (2013). https://goo.gl/Wkdt5P
[11] GnuCitizen. 2008. Name (mdns) poisoning attacks inside the lan. (2008). https://goo.gl/tvB7nB
[12] Google. 2017. Sign in using App Passwords. (2017). https://goo.gl/JxoseS
[13] Stephen T. Kent. 1993. Internet Privacy Enhanced Mail. *Commun. ACM* (1993). https://goo.gl/vRjxQ4
[14] United States Department of Homeland Security. 2017. Best Practices For Using Public Wi-Fi Tip Card. Brochure. (2017). https://goo.gl/QxuANv
[15] National Institute of Standards and Technology. 2017. Authenticator and Verifier Requirements. (2017). https://goo.gl/RQbiUr
[16] Alberto Ornaghi, Marco Valleri, Emilio Escobar, and Eric Milam. 2001. Ettercap. (2001). https://goo.gl/X6vzn8
[17] Symantec Security Response. 2016. Mirai: what you need to know about the botnet behind recent major DDoS attacks. (2016). https://goo.gl/Dm66by
[18] O. M. Ritchie and K. Thompson. 1978. The UNIX time-sharing system. In *The Bell System Technical Journal.* https://goo.gl/hU2SPr
[19] Franziska Roesner. 2017. Designing Application Permission Models that Meet User Expectations. *IEEE Security & Privacy* (2017). https://goo.gl/JfMM1c
[20] 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
[21] 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
[22] Jennifer G. Steiner, Clifford Neuman, and Jeffrey I. Schiller. 1988. Kerberos: An Authentication Service for Open Network Systems. (1988). https://goo.gl/Rj8GQm
[23] Matt Undy, Matt Bing, Aaron Turner, and Fred Klassen. 2013. Tcpreplay: Pcap editing & replay tools for *NIX. (2013). https://goo.gl/8AB5hq
[24] Primal Wijesekera et al. 2015. Android Permissions Remystified: A Field Study on Contextual Integrity. In *24th USENIX Security Symposium.* https://goo.gl/DhMfeB
[25] WikiLeaks. 2017. Vault 7: CIA Hacking Tools Revealed. (2017). https://goo.gl/1a1yS8
[26] 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

A DREAMCATCHER RULE TYPES

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, meaning 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 [11], 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.

Additionally, we have slight variants of these two rule types for broadcast packets and service discovery query packets, respectively.

B EXPERIMENTAL NETWORK DEVICES

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)	User's compromised laptop
Desktop1	Custom (Debian)	User's desktop
Phone1	Nexus 5x	User's phone
Printer1	Brother DCP-L2540DW	User's printer
Roommate1	Dell Optiplex (Windows 10)	Malicious roommate's desktop
Lightbulb1	Raspberry Pi (Raspbian)	Compromised light bulb bridge

C SURVEY PARTICIPANT PROFICIENCIES

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

D MECHANICAL TURK SURVEY SCENARIOS

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 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.

E LATENCY EXPERIMENTS

For **direct** packets, we send ICMP echo (ping) requests from the desktop to laptop. 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. 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* [23] 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 is 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 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.