Disrupting Attacker Value Propositions in Residential Networks

by

Joseph Turcotte

A Thesis Proposal

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the

Degree of Master of Science

in

Computer Science

by

May 2021

APPROVED:

Professor Craig A. Shue, Thesis Advisor

Professor Mark L. Claypool, Thesis Reader

Professor Craig E. Wills, Head of Department

Abstract

Attacks on residential networks continue to rise because poor security practices provide access for infiltration and compromised smart devices serve as ideal systems for botnets. Reducing the target on home networks requires a defense model that disrupts an attacker's value proposition; that is, the defense increases the required work to access a home network and decreases the perceived benefits of controlling home-networked devices.

In this work, we explore two classes of home network security, namely remote access and residential proxy detection. Remote access is a simple and straightforward approach that addresses the threat of home network intrusion and allows homeowners to access their devices and services anywhere outside the network. With a remote access scheme in place, attackers must spend more time and effort to find and compromise vulnerable devices. Residential proxy detection addresses the threat of IoT malware that allows attackers to remotely control home-networked devices; furthermore, residential proxies are relatively new services, and thus detecting them has not received much attention in network security research. With a proxy detection scheme in place, attackers' benefits decrease because they lose control of home-networked devices once their activity is detected.

We implement both classes of security services on a consumer-grade home router to show how the services work in practice. Additionally, we run performance evaluations to determine whether the router can run the services without significant performance overhead. Overall, we find that the service designs successfully address relevant attack vectors while offering minimal performance overhead for consumer-grade routers.

Acknowledgements

My thesis report represents the culmination of my studies at WPI, and so I thank my family, friends, peers and professors for being a part of my academic journey. In particular, I would like to thank Craig Shue for his role as my thesis advisor, Beckley Schowalter for her guidance and support in the CyberCorps Scholarship for Service program, Curtis Taylor for his role as my research mentor, Lorenzo De Carli for his role as my undergraduate capstone advisor, and Mark Claypool for his role as my thesis reader.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

While Internet-connected devices in home networks may offer greater convenience to homeowners, they also pose risks to security and privacy: adversaries can access these devices without authorization and use them to amplify the power of cyber-based attacks and hide malicious activity. SonicWall reported 32.4 million IoT malware attacks across the world from January to September 2020, representing a 30 percent increase from the previous year [1]. In many cases, attackers target smart devices in residential networks and commit privacy violations by accessing sensitive user data. The infamous Mirai botnet took control of smart televisions and gaming systems by exploiting default credentials to access these devices [2]. In this scenario, home networks abetted attacks on other networks through the botnet's command and control infrastructure; like a contagious virus, one compromised machine infects more devices until entire networks are under an attacker's control.

Attackers primarily target home networks because the benefits of controlling a device in a home network far outweigh the time and effort required to obtain network access. Attackers often consult documentation for common vulnerabilities to identify weaknesses in home network configurations and develop simple attack scripts to exploit these weaknesses. Poor security practices such as using default passwords and neglecting to apply security updates expand the attack surface and make some attacks easier to implement. If the initial compromise is successful, an attacker can move laterally throughout the network by identifying other vulnerable devices via internal network scans. With minimal research and development, an attacker can issue their own commands to smart devices and spy on homeowners by observing their network traffic.

To reverse the trend of increasing attacks on home networks, the networks must become less desirable targets: namely, the cost of attacking a home network must exceed the benefit. Assuming vendor solutions are unavailable, an alternative solution is to reuse existing infrastructure, perhaps in a way that the infrastructure was not originally created for. Smart devices often have closed-source APIs and may be impractical to modify; however, home routers can be flashed with firmware images to become programmable. Using a flashed router, it is possible to develop and run custom-made security services to address gaps in home network security, whether vulnerabilities are publicly-known or relatively obscure. A router with out-of-the-box services that defend from internal and external attack vectors compensates for poor security practices and is adaptable to the homeowner's needs.

In this paper, we explore the potential to deploy security services on a home router. While the range of services is theoretically unbounded, we focus on two case studies that embody the notion of disrupting attacker value propositions; that is, we suggest how an attacker's costs and benefits change with these services in place. First, we deploy remote access schemes to block external reconnaissance efforts while allowing authenticated users to gain remote access to the home network. The goals of our access schemes are to increase the barrier to entry for attackers and to maintain a low barrier of entry for legitimate parties. Next, we propose a residential proxy detection scheme that monitors network traffic for patterns that resemble proxying behavior. If we assume that an attacker has compromised a home-networked IoT device and is using it as a residential proxy, our goal is to cut off the attacker's access to the device and eliminate their benefits of controlling the device. We implement the required infrastructure to show how each service works and how we meet the goals of each service. We also evaluate performance overheads on the client and router endpoints to determine if it is practical to integrate these services into home networks. We find that our schemes successfully address relevant attack vectors while offering minimal performance overhead for consumer-grade routers.

The rest of the paper is outlined as follows: Chapter 2 presents background information on the unique properties of residential networks, the types of attacks that are possible, and the relevant defense models for these attacks. Chapter 3 provides implementation details for each security service discussed in the paper, and Chapter 4 describes how the services are evaluated. Chapter 5 presents experimental results and their implications. Finally, Chapter 6 discusses limitations of our work and areas of future work while offering concluding remarks.

Chapter 2

Background

The configuration, operation, and device profile of home networks share some similarities with enterprise networks; however, home networks also possess unique properties that warrant a separate behavioral analysis. These properties give rise to various security threats and possible solutions to manage these threats.

2.1 Residential Network Threats

With respect to network management and administration, corporations employ groups of information technology professionals and experienced administrators to configure and monitor enterprise networks. On the other hand, home networks often do not have a designated administrator. Even for residences with a designated lead, the administrator may have limited experience to focus on network security. The DHS Cybersecurity and Infrastructure Research Agency, or CISA, found that many homeowners hold misconceptions about their home networks; namely, their networks are too small to be at risk and their devices are 'secure enough' out of the box [3].

The belief that commercial products have enough security applies especially to home routers. While many corporate networks are equipped with high-powered and state-of-the-art routers, homeowners tend to purchase a consumer-grade router from their Internet service provider or use one that is outdated. Although these routers usually provide a reliable connection to the rest of the Internet, the security services on these routers are lacking and may even contain vulnerabilities. The 2020 Home Router Security Report found critical security flaws in all 127 routers analyzed, with an average of 53 critical-rated vulnerabilities [4]. When security patches for firmware become available, homeowners rarely install them due to time and experience constraints. Additionally, many homeowners do not change the default password provided with their routers; in a 2018 survey of 2,200 UK adults, 82 percent claimed that they did not change the default administrator password, and 48 percent of those respondents did not know why changing the password was necessary [5]. Because the default password is often shared across all routers of a brand or model range, it becomes easier for an external party to gain unauthorized access and target the devices connected to the network.

Residential networks typically contain several Internet of Things (IoT) devices created for domestic use. While corporate networks may allow these devices in office space as part of a "bring your own device" policy, the process to register a device to the company network can be onerous due to restrictions on permitted devices and their behaviors. IoT devices provide convenience and entertainment features; for example, smart speakers play music upon voice activation, and smart thermostats monitor room temperature and humidity. Even devices like smart egg trays, which alert a homeowner when store-purchased eggs have gone bad, can improve domestic life. Internet-connected devices have become a critical component of U.S. households; Kumar et al. conducted a global analysis of IoT adoption and found that more than 66 percent of U.S. homes had at least one connected device in 2019 [6]. However, researchers have discovered alarming vulnerabilities on many of these devices that could allow an external party to gain unauthorized access and obtain sensitive user data, such as full names, emails, and phone numbers. Kumar's analysis found that `telnet`, a communication protocol that has no built-in data protection and is vulnerable to eavesdropping, was supported on approximately 7 percent of IoT devices [6]. In a separate work, Valente et al. [7] discovered vulnerabilities in child smart toys that would allow an attacker to control the devices and speak directly to the child.

Finally, enterprises and home networks tend to differ on the choice between a symmetrical and asymmetrical network connection. A symmetrical connection provides equal bandwidth for upload and download traffic, and an asymmetrical connection partitions bandwidth for most efficient usage, particularly when one kind of traffic pattern is more common. Enterprises often choose a symmetrical connection to achieve bidirectionally fast data speeds; however, home networks most often opt for an asymmetrical connection. In addition to being cheaper, the asymmetrical model suits residences better because many end users download much more traffic than they upload [8]. For example, a homeowner communicating with a streaming service uploads small requests for content, and the service responds with large bursts of data for the user to download. While this distinction may seem trivial, asymmetrical communication patterns become important when considering anomalous behavior in residential networks, such as observing a suspiciously high volume of upload traffic.

2.2 Attacks on Residential Networks

The threats described in the previous section make home networks desirable targets for external observers and attackers with a foothold in the network. An attacker can utilize reconnaissance tools to learn more about home network properties and vulnerabilities, and a compromised device enables malicious activity on the network that is easier to obfuscate.

2.2.1 Port Scanning

Port scanning is a specific application of network scanning that legitimate and malicious parties use to probe a network and discover services that are open for communication. A port scanner does this by enumerating the sequence of possible port numbers and sending a SYN packet to each port on a networked device. Through this method, a party can determine which ports are open and closed by analyzing the response from the device. Port scanning constitutes one of many reconnaissance strategies where the goal is to develop an attack plan based on gathered information. Tools such as *nmap* automate network scans and make them easier to perform; given that scanning a single host takes a few seconds [9], scanning a small network is a low-cost reconnaissance activity that requires little technical expertise.

Home networks are not immune to port scanning efforts, although the collected information differs based on the adversary's placement. External observers can only learn about port forwarding rules that a home router designates, as home routers run Network Address Translation (NAT) and firewalls by default and prevent the outsider from performing an internal network scan. However, an attacker that compromises a device, perhaps through a successful phishing attack, has much greater capabilities. An attacker can conduct a full network scan originating from the compromised device to obtain a complete picture of the home network profile, including the types of IoT devices in the home and personally identifiable information associated with these devices. Although a port scan is relatively simple to detect via traffic analysis, homeowners who do not conduct regular network analyses are unlikely to notice this behavior.

2.2.2 Residential Proxies

Attackers may want to use home-networked devices to stealthily tunnel malicious traffic rather than conducting noisy network scans. An attacker may use a residential proxy device, which may be a home computer, an IoT device, or even a mobile phone [10]. When a party sends their traffic to the proxy, a completely new packet is created that originates from the proxy and is sent to the destination. Thus, the destination believes that the traffic originated from the proxy and not the original party. This approach allows attackers to evade blocklists and server-side firewalls that trust home network addresses more than network addresses from data centers or unknown locations. We note that the header contents and total size of proxy packets may differ from the original packets, as the proxy may add or remove TCP options or application-layer headers. However, the proxy does not modify packet payloads unless the proxy advertises this as part of its behavior.

Residential Proxy as a Service (RPaaS) providers such as Luminati and Oxylabs offer home-networked devices as proxies and claim that these devices join willingly; however, recent work in the domain has discovered that these devices are often used for malicious purposes and are likely compromised. Mi et al. [11] rented devices from these providers to examine their behavior and found evidence of malware hosting and phishing attempts. In a separate effort, Hanzawa and Kikuchi [12] examined RPaaS hosts located in Japan; they found that most hosts were laptops and that the most common malicious activity was port scanning. The behavior of residential proxies differs starkly from expected usage in the home. In fact, many of the proxy devices themselves have been blocklisted due to tunneling malicious traffic. Choi et al. [13] examine block list rates for RPaaS hosts in different countries, finding that 86 percent of residential proxies are prone to block listing. While this observation contradicts the claim that servers trust residential addresses more than data center addresses, the layer of obfuscation provides enough of an incentive to encourage proxy use. Although dropping proxy traffic removes a potential attack vector, it also hampers the usefulness of the device for legitimate purposes if many servers add the device to their blocklists.

Obfuscation models complicate security auditing and investigation, as it becomes more difficult to attribute an action to a specific party. Additionally, the party could successfully deny that they committed the action when attribution is impossible. In the context of law enforcement, investigators often traverse multilayered proxy networks to find the true originator of a malicious request or action. The 2016 Mirai botnet transformed infected IoT devices into proxy servers after hijacking 900,000 home routers [14]; the attackers were able to evade capture for several months thanks to concealing their actions with proxy devices. In addition, the use of Tor [15] in home networks allows personal computers and IoT devices to be configured as relay nodes that behave similar to residential proxies in principle. Tor traffic is designed to evade detection; this property benefits users in locations where censorship is common, but also allows malicious traffic to travel covertly between a source and a destination. The lack of visibility into home networks further complicates attribution analysis, as it is not feasible for an outside party to determine whether a device exhibits proxy-like behavior. If left unchecked, attackers will inflict harm on the networks that grant them increased stealth.

2.3 Defense Models for Residential Networks

Given the attack vectors that pose harm to home networks, it is important to minimize the impact and availability of these vectors. The goal of a defense is to decrease the benefit of attacking while increasing the cost of attacking and the risk of detection. The cost-benefit analysis an adversary undertakes is part of the adversary's value proposition; thus, any legitimate defense must cause enough disruption to the value proposition that the costs outweigh the benefits.

Fortunately, attackers are not the only parties that can leverage obfuscation techniques. One model that operates with the goal of disruption is termed 'needle in a haystack,' or haystack. According to this model, the effort to obtain a resource increases significantly beyond the value of obtaining the resource. While the haystack approach has some similarities with 'security through obscurity' (a generally discouraged approach to security), the haystack paradigm more appropriately satisfies the authentication property because a legitimate party can easily access the resource. This distinction provides the potential for authenticated parties to access their home-networked devices and services remotely.

We explore remote access mechanisms and residential proxy detection as vehicles of the haystack model. These services can work independently of each other or work in conjunction to offer a multilayered defense for residential networks. This paper also discusses potential extensions to the proposed services, such as leveraging cyber deception techniques and outsourcing security processing to the cloud.

2.3.1 Remote Access Mechanisms

The first goal of a remote access mechanism is to enable an authorized user outside the home network to gain access to an internal service. For example, a user traveling in a different country may need to copy files stored on their home server to their laptop; according to the remote access model, the user must perform an authentication procedure to establish their right to access and copy these files. The second goal is to drop unauthenticated requests for communication, thus maintaining a closed-door policy for parties that cannot prove their right to access a device.

Using the home server example, the traveling homeowner only needs access to a file transfer service to fulfill their need. Therefore, rather than granting a remote user complete access to a home-networked device, we specify access at the port level to follow the principle of least privilege. Once authenticated, the user would be able to contact the home server on port 21 to utilize the file transfer protocol. When communication ends, the home network may require re-authentication to access the service again or allow access to persist for a short period of time. It is important for any authentication scheme to resist replay attacks in which an on-path adversary observes the request packets and sends them again to gain unauthorized access. Given the volume of traffic that flows between an ISP and the home networks it services, adversaries typically resort to sampling a portion of the traffic. While the sampling constraint decreases the probability of viewing the authentication request, the adversary still has a chance to observe and replay the complete request; therefore, each remote access mechanism should make a replay attack impractical to perform successfully.

Several works have explored authentication mechanisms for remote access. Some of these methods utilize predefined access control lists, while others rely on user-provided capabilities. Isabel [16] discusses port knocking, a method in which a party issues requests to a series of port numbers; this sequence of numbers maps to a single port on a device, and the party gains access to that port upon success. Degraaf et al. [17] expand on traditional port knocking to handle complications with out-of-order packet delivery when the sequence order is enforced. The authors also address NAT complications in which multiple users share the same source address, so granting access to that IP address could give other users the same level of privilege as the user who initiated the request. Haber et al. [18] evaluate the feasibility of remote access for services inside a home network using virtual instances of these services, thus mitigating the impacts of a faulty authentication request on real hardware. Parno et al. [19] propose a computation or 'puzzle'-based system that reduces the impacts of denial-of-capability attacks. The authors use a router to process puzzle answers in a fairly allocated manner; this approach increases the chances that a legitimate user's authentication request makes it to the router, especially in the case where an attacker floods the channel to block the request.

Some capability-based methods reuse or modify existing infrastructure. Shue et al. [20] transform Domain Name System (DNS) resolutions into capabilities that clients can use to communicate with remote servers. While this scheme also faces NAT complications, the reuse of DNS infrastructure allows the process to remain relatively lightweight and practical. Taylor et al. [21] implement fine-grained access control for internal network flows that only requires standard software deployment tools, such as a configuration manager. In addition, network operators can write access control rules to permit or deny actions inside the network.

2.3.2 Residential Proxy Detection

If a device inside the home network is compromised, an attacker can direct the device to perform internal network scans or use the device as a proxy to forward malicious communication. Because residential proxies are relatively new compared to scanning techniques, there are fewer works available that explore defense models for residential proxies. It should be noted that intrusion prevention services (IPSs) can help defend against the initial device compromise; however, should an IPS miss a successful intrusion attempt, a detection mechanism can catch proxy behavior before the device inflicts serious damage on the home network.

One method to detect proxy behavior relies on fine-grained network traffic analysis; this approach relates to the asymmetry between upload and download rates in home networks. End users download at much higher rates than they upload; thus, a user uploading a large amount of content in a short amount of time is a possible indicator of proxy behavior. Of course, a burst of upload traffic does not sufficiently prove that a device exhibits proxy-like behavior, as a proxy must receive data flows from a source and forward the flows to a destination. The work by Mi et al. [11] discovers RPaaS hosts by leasing the hosts from providers and tracking the host behavior to discover additional hosts. While this work defined the global breadth of residential proxy usage, we explore whether home networks can perform their own analysis, rather than hiring a third party to determine if a compromise took place. Tosun et al. [22] implement network anomaly detection algorithms to identify RPaaS flows on compromised consumer devices. Their approach runs on the compromised host, which may present complications if an attacker remotely controls the device and identifies a suspicious process running on it.

Other works have examined fingerprinting techniques to detect proxy- or bot-like activity. Antonakakis et al. [23] perform feature analysis on DNS queries to distinguish malicious and agile use of DNS from legitimate use. Stalmans and Irwin [24] implement a similar classification model to identify malicious domains by examining DNS query responses at the network edge. With respect to anonymized traffic, such as traffic originating from a Tor browser, Carvalho et al. [25] examine network traffic at the session level to find TLS fields unique to Tor and create detection rules from these fields. While anonymized traffic is intentionally more difficult to detect, the authors showed that it is possible to identify unique features and patterns to effectively fingerprint this traffic.

Although related work has described typical use cases for residential proxies as nefarious, there are legitimate use cases that can benefit homeowners. Using the traveling example, a homeowner may want their online activity to appear as if it originated from their home, especially in cases of traffic discrimination against specific geographical regions. The work by Choi [13] finds significantly lower block list rates for U.S. proxies, compared to countries like Turkey and Indonesia. Additionally, open-source solutions such as nginx [26] provide server-side proxy capabilities to improve load balancing and overall performance. A homeowner could use a proxy inside their home network to reap these benefits for locally hosted websites, such as a personal blog or business page. It is possible to catch unauthorized proxies while allowing legitimate proxies by combining the remote access mechanisms and the proxy detection scheme into a multilayered defense. While this paper does not combine the two service classes, Section 6.1.1 elaborates further on this possibility.

2.3.3 Cyber Deception

In some scenarios, it is possible to convince the attacker that they have been granted access to the resource of interest, such as a locally-networked IoT device, without exposing the resource itself. Cyber deception, a defense practice that presents a host with seemingly genuine connections to devices and services, aims to increase the attacker's risk of detection while protecting legitimate resources. This practice is performed on decoy devices called honeypots or in a network of honeypots known as a honeynet.

Several works have explored the potential of deception techniques to enhance traditional security models. Han et al. [27] present a comprehensive characterization of existing deception solutions, including the advantages and limitations of each. In a separate work, Han [28] presents HoneyMix, a honeynet that matches each incoming connection with the most appropriate honeypot based on the connection context. Finally, Chiang et al. [29] deploy cyber deception in an enterprise network setting and propose a solution to the static nature of honeypot devices. The authors argue that dynamic adaptation of the network view is critical for preventing an attacker from detecting the honeypot.

It is difficult to evaluate the effectiveness of cyber deception techniques in academic environments, as realistic experiments would require a system that invites attackers to target. Furthermore, quantitative measurements for deception techniques can be ambiguous and may be specific to an individual attacker's behaviors. Sugrim et al. [30] propose a metric system that depends on an attacker's belief state: the level of confidence that they are interacting with a real service. The authors utilize a Bayesian inference model that updates the attacker's belief state over time as the attacker interacts with more devices in the network. This model could apply to a real-time deception system that monitors incoming connections and evaluates how the communicating party adapts their behavior upon interacting with a honeypot.

While these works have helped establish cyber deception as a promising defense model, there has been little discussion of how to apply deception techniques in home networks. Remote access coupled with cyber deception may offer greater access limitations for residential networks than it provides as a standalone feature. Besides offering access capabilities for legitimate users, unauthorized users are less certain about the success or failure of their connection attempts. With respect to residential proxy detection, cyber deception can convince the proxy that it is forwarding traffic to a real server; this technique could diminish the proxy's usefulness to an attacker and protect the device from widespread blocklisting.

2.3.4 Cloud Processing and Collective Security

When considering the placement of security services for home networks, one logical place to install the services is on the home router. The router has a 'home advantage' due to its complete visibility of the network; this visibility extends to the devices that exist in the network and the unique communication patterns of each device. Additionally, the router serves as the interface to the rest of the Internet, which allows it to work in conjunction with firewalls and NAT rules to filter out unwanted traffic. Sundaresan et al. [31] present the effective deployment of routers running custom software as part of the BISMark testbed project. Flashing a commercial router with custom firmware makes it possible to implement the security services described thus far. Simpson et al. [32] propose an in-hub security manager that runs security modules on top of the home router. These security modules could include remote access and proxy detection as part of the security manager's operations.

Homeowners who are concerned about the impact of locally run security services on their network's performance may consider outsourcing security processing to the cloud. The work by Sundaresan [31] discusses architectural constraints for deploying custom routers, such as limited client resources on commercial routers, which cloud computing could help overcome. Several works have demonstrated that cloud-based solutions can shoulder heavy computations without introducing large performance delays for home networks. Taylor et al. [33] explore performance impacts of outsourcing residential network security to the cloud; they found that 90 percent of end users sampled across the United States experienced acceptable performance with the outsourcing mechanism in place. In a separate effort, Taylor [34] leverages cloud-based middleboxes to monitor the Transport Layer Security (TLS) handshake process and flag inauthentic TLS servers to protect homeowners from visiting insecure web pages. Liu et al. [35] also redirect security maintenance to the cloud, but focus on authenticating endpoints with two-factor authentication to protect IoT devices in the home.

Rather than isolating defense models to individual home networks, it is possible for home networks to communicate with each other and amplify their defensive capacities. Liu et al. [36] propose a distributed approach to home network security in which a family of routers identify each other via broadcast messages and cooperate on network filtering. This approach would function like a neighborhood watch program, where each home network monitors suspicious connection requests and warns networks in its vicinity. Hunter et al. [37] explore a distributed approach to tarpitting, a mechanism that purposely delays incoming connections to discourage network abuse and spamming. Using Hunter's model, mail servers inform each other about misbehaving entities and slow the connection for those entities.

Finally, home network security can benefit from software defined networking (SDN), a paradigm that allows for dynamic, programmable network configuration. With SDN, it is possible to change network views on-the-fly and implement custom traffic analysis. Sivaraman et al. [38] introduce a scheme to dynamically quarantine IoT devices based on network activity and contextual factors, such as the time of day, and MacFarland et al. [39] implement a moving-target defense with SDN to limit the value of finding active IP addresses in home networks. For home networks, an SDN controller could make decisions for the entire network group from high-level traffic statistics, such as the amount of traffic entering or leaving the group. This not only allows for higher processing efficiency, but also for more secure networks to shield vulnerable home networks under a collective security umbrella. Section 6.1.2 explores the combination of SDN with residential proxy detection.

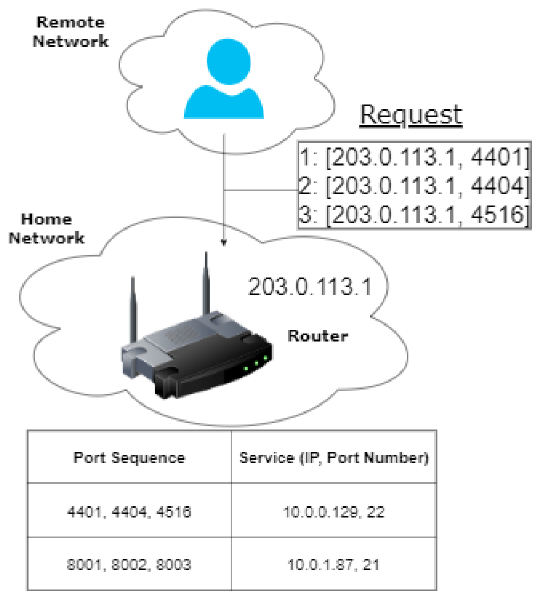
Chapter 3

Implementation

In this chapter, we present the service designs for remote access and residential proxy detection. Each service design includes the module to be placed and run on the home router, as well as the supporting infrastructure that demonstrates how the service functions. This chapter discusses the basic components of each implemented service and extensions to the fundamental scheme.

3.1 Port Knocking

As described in Section 2.3.1, port knocking represents a traditional remote access method to open external ports on a firewall that would otherwise remain closed. Figure 3.1 illustrates the scheme from the client and router perspectives; the client sends three packets to the router, where each packet is labeled with the router's address and the port to contact the router on. Upon observing an incoming packet, the router invokes an event handler that extracts and records information from the packet, including the source IP address and destination port. Next, the router consults a mapping of port sequences to services, where unique pairs of internal IP addresses and port numbers represent each service. If the destination port number appears in any mapping, the router records the seen port number with the source IP address. If the packet's destination port successfully completes a port sequence, the router creates a rule that allows the client to use the service by sending traffic to the given internal address and port number. The rule expires after a short amount of time, after which the sequence needs to be sent again to regain access. We note that the router does not respond to the client's request; rather, the client can send a new communication request to check whether the request succeeded.



Port Sequence	Service (IP, Port Number)
4401, 4404, 4516	10.0.0.129, 22
8001, 8002, 8003	10.0.1.87, 21

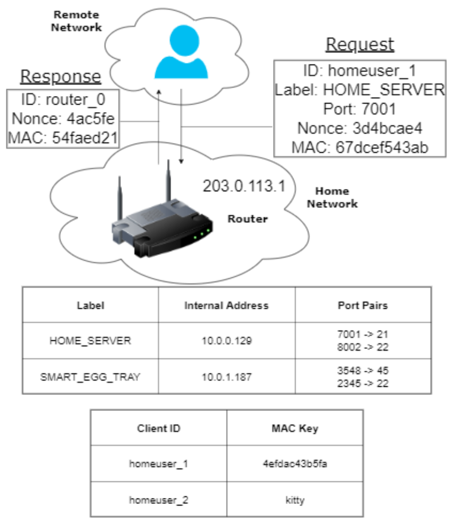
Figure 3.1: A router receiving and processing a three-port sequence from a remote client

While the basic scheme offers greater protection than leaving all ports open, an attacker could still gain access by guessing the correct sequence with a brute-force approach or by observing a valid sequence and replaying it. We extend the basic scheme to be covert on the client and router endpoints, thus making the aforementioned attacks less likely to succeed and riskier to attempt. At the client endpoint, we minimize the risk of obtaining a port sequence without authorization by embedding the sequence within a larger stream of port numbers, rather than sending the sequence contiguously. The client initiates a request to every router port specified in the stream; when the router processes these requests, it will ignore any that do not correspond to a legitimate sequence. This client-side behavior hampers the effectiveness of replay attacks, as the traffic sampling limitation described in Section 2.3.1 becomes more difficult to overcome when it is unclear which ports are part of the valid sequence.

At the router endpoint, the event handler maintains a list of 'alarm' ports that prompt a series of actions if a connection request is received at one of those ports. In our design, the router immediately expires all rules corresponding to the client that triggered the alarm port and drops subsequent communication requests from that client for a short time. The router also instructs a low-interaction honeypot to record the client's identity in a log file. The purpose of alarm ports is to discourage brute-force attacks that try every combination of port numbers to guess a valid sequence, since it is likely that the client will eventually hit an alarm port and be blocklisted. Additionally, it is possible to further mitigate the impact of replay attacks by periodically rotating the sequence-to-service mappings or utilizing one-time sequences; this concept is explored in Section 3.3.

3.2 Explicit Network Request

Like port knocking, an explicit network request (ENR) protocol offers a method for remote access with some key differences. Rather than sending a stream of packets to the router with an embedded port sequence, the client sends a single packet containing the authentication request. Figure 3.2 illustrates the flow of events for ENR. When a client wishes to gain access to an internal service, the client sends a message containing a unique client identifier, a unique name for a service inside the network, a public-facing port number, and a nonce value to guard against replay attacks. To preserve data integrity, the client also appends a message authentication code (MAC) digest computed with a symmetric MAC key that the router and client negotiate on out-of-band. The router listens for incoming authentication requests and delegates processing to an event handler that consults two tables stored on the router to process the request. The first table contains a mapping of service names to internal device addresses and port numbers, allowing the event handler to determine which internal service address and port the client wants to access. The second table contains a mapping of unique client identifiers to symmetric MAC keys;



Label	Internal Address	Port Pairs
HOME_SERVER	10.0.0.129	7001 -> 21 8002 -> 22
SMART_EGG_TRAY	10.0.1.187	3548 -> 45 2345 -> 22

Client ID	MAC Key
homeuser_1	4efdac43b5fa
homeuser_2	kitty

Figure 3.2: A home router receiving an authentication request from a remote client using ENR

this table allows the event handler to generate a MAC digest for the received message and compare it to what the client sent. The event handler verifies the request if the client's identity appears in the router's tables and the MAC digests match. ENR also makes use of alarm ports to block any client who triggers an alarm port in their authentication request.

After the event handler completes the verification procedure, the router explicitly responds to the client to indicate whether the authentication request succeeded. In its response, the router sends its own unique identifier, a separate nonce value, and a 0/1 binary value to represent failure or success, respectively. However, this information may be useful to an on-path adversary who wants to learn which clients were successfully authenticated. We require a computation on the client endpoint to determine whether the request succeeded without requiring the router to send the message in its response. The router does this by computing the MAC digest with the binary value, removing the binary value, and appending the digest to the response. The client computes both possible digests to infer the router's response.

ENR and port knocking accomplish the same goal, yet each protocol's inner details differ enough to warrant a comparative analysis. We compare the protocols with respect to service reachability, protocol detectability, and connection establishment complexity and present our findings in the Results chapter.

3.3 DNS Credential Rotation

Port knocking and ENR provide means for a client to perform remote authentication. However, these methods require the client to possess credentials such as a port sequence or an internal device name beforehand. In the event that these credentials change periodically, there must be supporting infrastructure for clients to obtain the most recent version. We leverage existing DNS infrastructure to allow clients to retrieve DNS records on-the-fly and use them for remote access. Figure 3.3 illustrates the basic scheme: the client must know a unique domain that maps to the desired home router's network address. The client performs a DNS lookup for that domain, and the client's resolver eventually reaches an authoritative DNS server that answers for that domain. The DNS server responds with two records; the first is an A record that provides the IP address of the home router, and the second is a TXT record that provides the credentials for the client to present to the router. Should the client choose to use port knocking in tandem with DNS, the TXT record will contain the port sequence for the client to send to the router. For ENR, the TXT record will contain the service's name and public port to embed in the client's authentication message. Once the client receives the authoritative server's response, the authentication procedure proceeds as described in Sections 3.1 and 3.2.

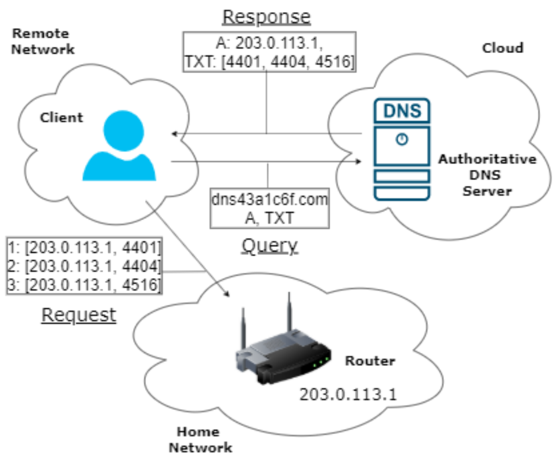


Figure 3.3: A client querying an authoritative DNS server for authentication credentials

DNS offers a lightweight mechanism to rotate credentials for home network access, and there are several viable options to handle rotation. One option is for the router to send periodic updates to the authoritative server with new records to provide to clients that query the server. No additional entities are required for this approach, but the router is responsible for generating and sending new records. Another option is to create a subzone authority within the home router that the authoritative server queries to get the required records. This option does not require outgoing messages from the router. In addition, the subzone authority securely stores new records if the authoritative server is compromised. However, the authoritative server must issue a query inside the network every time a client performs a lookup. Each service option has its benefits and drawbacks based on the individual properties of the network that implements the service.

3.4 Residential Proxy Detection

Before explaining the algorithm for detecting residential proxy behavior, we define the set of indicators on which the algorithm relies. As described in Section 2.2.2, residential proxies are primarily responsible for forwarding traffic from a source to a destination by repackaging the traffic to originate from the proxy, rather than the source. Therefore, a router must observe data from a source going into the proxy and later exiting the proxy towards a destination, where the source and destination are distinct. Additionally, we assume that the proxy will forward traffic as quickly as possible, rather than intentionally delaying client traffic. Therefore, the router must also observe a temporal correlation between the ingoing and outgoing traffic in which ingoing packets are almost immediately forwarded as outgoing packets. Finally, based on the properties of residential proxies discussed in Section 2.2.2, we assume that the packet contents are not significantly modified when being forwarded. This assumption implies that the router should observe a spatial correlation in which the packet sizes of ingoing packets are nearly identical to that of outgoing packets. While it is possible to use deep packet inspection methods to examine packet payloads for evidence of proxy behavior, this approach is unlikely to work on encrypted traffic and is more computationally intensive than relying on packet metadata. Therefore, we exclude payloads as proxy indicators.

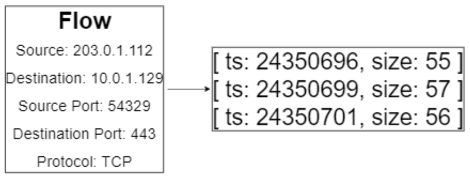


Figure 3.4: Definition for a flow and the packets associated with the flow

The proxy detection algorithm performs traffic analysis to 'score' pairs of flows based on the indicators described above. Figure 3.4 illustrates the definition of a flow, which includes the source and destination IP addresses, the source and destination ports, and the transport layer protocol. Each flow maps to a list of packets that match the flow properties; the router records each packet's time stamp and size and adds it to the list of observed packets for that flow. Algorithm 1 provides the pseudocode for the routine that performs pairwise flow comparisons. Flow pairs that do not have a common relay, or only involve two entities, are given a score of zero since they do not meet the first indicator for proxy behavior. Next, the algorithm compares the flow pair packets in chronological order. It utilizes two tolerance values to determine whether the packets follow the second and third indicators, with T as the maximum allowed difference in timestamp values and S as the maximum allowed difference in packet sizes. Finally, the algorithm computes the score for the flow pair as the number of packet matches divided by the maximum number of possible comparisons (the higher value between the number of ingoing and outgoing packets). The value of the denominator compensates for pairs with significantly different packet counts, thus encouraging flows with roughly equal packet counts to receive a higher score.

Algorithm 1: Pairwise flow comparison algorithm

```
SET scores to empty list;
for each pair of flows f,g do
    if f.dst-ip == g.src-ip AND f.src-ip != g.dst-ip then
        SET matches to 0;
        for each pair of packets p,q in f.packets, g.packets do
            if $q.ts - p.ts \leq T$ AND $abs(q.size - p.size) \leq S$ then
                SET matches to matches + 1;
            end
        end
        SET score to $matches \div max(len(f.packets), len(g.packets))$;
        ADD score to scores;
    else
        ADD 0 to scores;
    end
end
```

We note that the algorithm alone does not produce a definitive answer for whether a device is a proxy. Rather, the algorithm ascertains which flow pairs follow the indicators that imply forwarding behavior. The next step is to analyze flow pairs with high scores and identify which address serves as the relay in the communication. The probability that a device acts as a proxy increases with the number of high-scoring flow pairs involving that device; the router may choose to act on a high probability by blocking flows from all sources in the high-scoring flow pairs or by quarantining the suspected proxy device for a short time. As a final note, the algorithm can operate offline or on live network communications. In the offline mode, we extract flow data from a packet capture and pass the data to the algorithm for scoring. In the live mode, the router records all ingoing and outgoing communication for a short window and calculates scores for all flow pairs observed in that window. Then, the router clears all observed flow data and begins recording communication for the next window. We further elaborate on these modes of operation in the Evaluation and Results chapters.

Chapter 4

Evaluation

A home network defense must offer minimal performance overhead and provide the same, if not greater, level of user privacy as a traditional network architecture. This chapter outlines the metrics that we use to qualitatively and quantitatively evaluate each defense, as well as the experimental setups that allow us to obtain measurements for those metrics.

4.1 Software and Hardware Tools

Since the home router performs many of the critical computations required for the security services to function properly, we carefully choose the firmware and hardware components with respect to suitability for residential networks. First, the choice of firmware is version 19.07 of `OpenWrt`, an open-source Linux distribution designed for embedded devices [40]. `OpenWrt` provides shell access, allows for quick configuration changes, and supports SDN protocols. Next, the TP-Link Archer C7 AC1750 router serves as the home router for experiments, as it is compatible with the `OpenWrt` distribution and is commonly marketed to homeowners. We flash the router with an `OpenWrt` firmware image and configure the router as an Internet access point.

The experimental testbed relies on various software libraries and hardware components. We develop each security service as a standalone module in version 3.7 of the Python programming language. The modules utilize several built-in and external libraries, including: `scapy` for network traffic capture; `dpkt` for offline packet capture parsing; `socket` for establishing network connections between testbed entities; `time` for recording wall-clock time; `hmac`, `hashlib`, and `uuid` for generating MAC digests and nonce values; `random` for generating random values; `dns` for performing DNS lookups; and `subprocess` for creating firewall rules that allow remote access. The TP-Link router includes a USB port that helps address the router's memory limitations. We install the Python 3.7 distribution and the required libraries on a USB drive and connect it to the router such that the modules can run from external storage.

Finally, we build supporting infrastructure for the remainder of the experimental testbed. Except for the router, a unique virtual machine represents each entity in the testbed, including the authoritative DNS server, internal resource, proxy device, and remote clients. Some of the virtual machines run as Microsoft Azure cloud instances, and others run inside the university network. We also develop Python scripts for each piece of supporting infrastructure and run each from the appropriate entity. We use `BIND9` to configure the authoritative DNS server by following the documentation from [41].

4.2 Remote Access Evaluation

We measure wall-clock delay as the primary metric for performance overhead. Home networks are unlikely to adopt mechanisms that slow down overall network speed, as performance and quality of service are critical for use cases such as video conferencing and online gaming. In the following sections, we consider the predominant computation that contributes to the delay, how often the entity performs the computation, and how we measure the delay.

4.2.1 Port Knocking

The setup to measure computation delays on the client and router endpoints largely follows what is outlined in Figure 3.1. First, we start a socket listener on the honeypot module so the router can establish communication with the honeypot. Next, we start the router module on the TP-Link router; the module reads in the port sequence mappings and list of alarm ports from files, establishes a socket connection with the honeypot, and begins listening for incoming packets. Then, we begin the client module by embedding the port sequence supplied via the command line within a list of randomly generated port numbers. The client sends one UDP packet to the router's IP address for each port in the list; while the client sends packets, the router module processes each packet by checking its sets of alarm ports and valid sequence mappings. At the end of the experiment, we stop the router and honeypot modules and check the list of `iptables` rules to verify the success of the client's authentication request.

We measure delay on the client and router endpoints by following four steps: (i) recording the wall-clock time just before the core computation begins, (ii) executing the core computation, (iii) recording the wall-clock time just after the core computation completes, and (iv) calculating the difference between the recorded times. Therefore, we require a precise definition for the core computation on each endpoint. For the client, the core computation includes generating a stream of port numbers and sending a packet to the router for each port number in the sequence. We note that the size of the port sequence is an independent variable in the experiment; for consistency across experiments, we hold this variable constant at 256 port numbers. The core computation for the router is more complex, as the action the router takes for each packet relies on the packet contents and may require updating firewall rules if an alarm port is hit or if a port sequence is completed. The experimental setup addresses the router-side complexity by reporting a maximum computation delay along with the average delay for the entire sequence; Section 5.1 further elaborates on the significance of these measurements. To obtain statistically sound performance results, we run the experiment thirty times and log computation delays each time. The Results chapter reports the average delay value and standard deviation, along with the maximum delay value on the router endpoint.

4.2.2 Explicit Network Request

The setup to measure computation delays on the client and router endpoints largely follows what is outlined in Figure 3.2. We begin the experiment by starting the router module on the TP-Link router. The module reads in service mappings and symmetric MAC keys from files and starts a socket listener to await incoming client connections. Next, we begin the client module by constructing its authentication message and MAC digest and sending the message to the router. The router module follows the verification procedure described in Section 3.2, constructs its response and MAC digest with the same MAC key, and sends the response. The client processes the response and reports the success or failure of the request. At the end of the experiment, we stop the router module and check the list of `iptables` rules to verify the success of the client's authentication request.

We measure delay following the same steps outlined for port knocking. For the client, the core computation includes generating the authentication message and MAC digest for the appropriate internal service, receiving and parsing the router's response, and computing MAC digests to determine whether access is granted. The core computation for the router includes parsing a client request, verifying the request by computing a MAC digest, and generating its own message and MAC digest to send back to the client. Like the setup for port knocking, we run the experiment thirty times and log the computation delays each time. We also report the maximum delay value along with the average delay value and standard deviation because the router actions vary based on the contents of the authentication request.

4.2.3 DNS Credential Rotation

The setup to measure computation delays on the client endpoint largely follows what is outlined in Figure 3.3. We begin the experiment by restarting the `BIND9` service on the authoritative DNS server to accept new client queries. Because the client delay does not rely on periodic rotations of DNS records, the records remain constant across experiment runs. Next, we begin the client-side DNS module which requests A and TXT records from the authoritative server. Once the client receives the records, the client parses the A and TXT records in the server's response and reports the values it obtained from the records. For experiment completeness, the client performs one of port knocking or ENR using the information from the DNS records; then, we check the router's list of `iptables` rules to verify the legitimacy of the DNS server's response.

We treat the DNS extension somewhat uniquely because it can utilize either of the two aforementioned schemes to deliver the authentication request. Therefore, we assume that the overheads for those schemes are part of the DNS extension and do not remeasure them. The only new computation occurs on the client side and includes issuing a DNS query and processing the answer from the authoritative DNS server. We note that the placement of the DNS server is important when measuring delay; we host the DNS server on a virtual machine in the US, so the client's geolocation impacts the total delay on the client endpoint. However, we use the same virtual machines when evaluating each remote access scheme to negate the impact of placement in our measurements.

Like port knocking and ENR, we run the experiment thirty times and log the computation delay each time. The Results chapter reports the average delay value and standard deviation for the client endpoint.

4.3 Residential Proxy Evaluation

We present two live experimental setups to evaluate the residential proxy detection algorithm. Offline analysis can support or refute the results from live analysis or help when live analysis is not possible. In addition, we evaluate the algorithm's runtime to determine if the incurred overhead is feasible for home networks.

4.3.1 Algorithm Runtime

The core computation for the detection algorithm is outlined in Algorithm 1 from Section 3.4. The outer loop will always run in $O(N^2)$ time, because we score every permutation of ingoing and outgoing flows in a pairwise fashion. With respect to the inner loop, the worst case runtime occurs when no packets match between the ingoing and outgoing flows; due to the pairwise comparison behavior of the packet matching step, the worst-case runtime of the inner loop is also $O(N^2)$, yielding a worst-case runtime of $O(N^4)$. However, the worst-case is very unlikely to be achieved in practice. First, we expect that many flow pairs do not satisfy the primary forwarding indicator because the ingoing destination address and outgoing source address do not match. Therefore, the algorithm short-circuits these pairs before packet-based comparison happens. Second, because we record packets in chronological order, a match likely occurs right away or not at all; a high scoring flow pair biases the runtime towards $O(N)$, whereas a low scoring flow pair is closer to $O(N^2)$. Nevertheless, we note the importance of evaluating the algorithm's runtime under various network traffic conditions to determine if a home router can run the computation feasibly.

We develop runtime experiments based on two independent variables that most significantly impact the runtime measurement: (i) the number of unique flows in the dataset, and (ii) the number of packets per flow. The first experiment uses increasingly high numbers of unique flows and relatively low packets per flow (50, 100, 250, and 500 unique flows, and 5, 10, 20, and 40 packets per flow). This experiment resembles a home network scenario where multiple users are browsing different websites and receive short replies with content. Alternatively, these traffic conditions may imply that a device is acting as a public proxy for dozens of clients at once. The second experiment uses increasingly high numbers of packets per flow and relatively low unique flows by swapping the independent variable values from the first experiment. This experiment resembles a home network scenario where devices receive large bursts of content from a streaming service or online gaming platform. Alternatively, these traffic conditions may imply that a device is acting as a proxy for one client that wishes to upload or download large amounts of data from a few different servers. We consider both workloads separately to see if one computation dominates the algorithm runtime.

The experimental setup to evaluate the algorithm's computational overhead uses simulated data rather than real network traffic, since the algorithm can operate independently of a home network architecture. We create a simple Python class for a flow following the definition from Figure 3.4 and instantiate flow objects from that class. We represent packets for a flow as a list of timestamp and packet size pairs. We note that when we create the flow objects and packets for each flow, we hardcode source and destination IP addresses such that a subset of flow pairs will move onto the packet comparison stage to avoid trivial runtime results. We also hardcode timestamp and packet size values that allow for some matches between packets. After creating the required number of flow objects and packets for each flow, we pass the flow data to the algorithm. The algorithm records the current wall-clock time as the start point of computation, runs through its core computation as described above, logs the amount of time that passed since the start point of computation and finishes. We repeat this experiment thirty times with the same number of flows and packets per flow; then, we compute the average and standard deviation across the collection of measurements. Additionally, we run the experiment with different values for each independent variable as described above.

4.3.2 Live Analysis

The first experiment for live analysis involves one party using a residential proxy to tunnel TLS traffic. This setup is most similar to a client forwarding traffic through a Tor relay node to maintain anonymity. Figure 4.1 illustrates the experimental setup for TLS tunneling; the blue arrows represent a source sending network traffic to the residential proxy device, and the red arrows represent the proxy forwarding the source's traffic to the destination. Each entity in the diagram requires its own module as part of the experimental setup. The source and destination, represented as cloud-based virtual machines, run Python modules that establish socket connections with the proxy device. For consistency across experiments, we hold the rate at which the source sends messages at 4 packets per second. Next, a laptop inside the home network representing the proxy device runs a Python module that listens for messages from the client and forwards those messages to the destination. Finally, the TP-Link router runs a Python module that contains the detection algorithm and code that generates flow data from live network traffic.

The second experiment for live analysis involves one party sending an HTTPS request to a locally-networked server through a server-side proxy. This setup is most similar to a server-side nginx proxy that a homeowner configures inside their home network. Figure 4.2 illustrates the experimental setup for HTTPS communication. The main difference from the first setup is the placement of the party that the client interacts with: rather than communicating with an outside party, the client requests content that exists inside the home network. Like the TLS tunneling scenario, each entity in the diagram requires its own module as part of the experimental setup. The client runs a Python module that establishes a socket connection with the proxy device and submits HTTPS requests to obtain content from inside the home network. The home server runs a Python module that receives HTTPS requests from the proxy device and sends the desired resource back to the requester through the proxy. We use the same modules for the router and proxy device as the ones in the TLS tunneling setup.

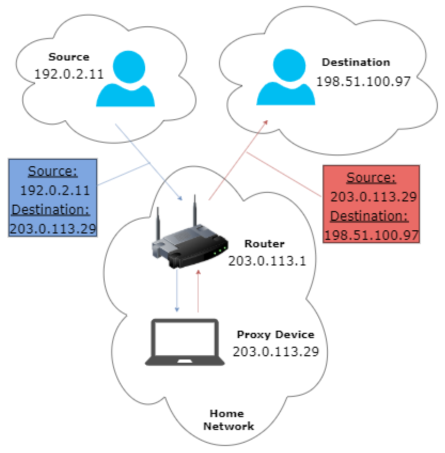


Figure 4.1: The flow of communication for the TLS tunneling setup

In each experimental setup, we run the router module first to listen for network traffic; then, the client begins sending messages to the proxy. Based on the implementation details from Section 3.4, the router module runs the detection algorithm to calculate scores for the observed flows in that window and logs these scores to a file. After running the algorithm, the router clears all flow data from the window and continues observing network communication. We run the experiment for thirty windows to obtain statistically sound results and compute the average score and standard deviation from the score measurements.

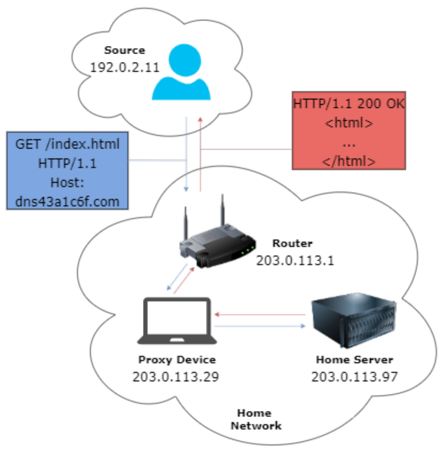


Figure 4.2: The flow of communication for the HTTPS request setup

We measure the algorithm's performance under perfect and imperfect forwarding conditions. We modify the proxy device to adhere to the following principles: (i) the proxy delays forwarding each packet by a random value measured in seconds; (ii) the proxy changes the packet size by a random value measured in bytes; and (iii) the proxy drops a packet with a probability in the range of [0,1]. Under perfect conditions, packets are forwarded immediately to the destination without being modified and there is no risk of packet loss. Therefore, we expect the score for the pairs of flows in the figures to be roughly 1.0. Under imperfect forwarding conditions, we expect the score for the flow pairs in the diagram to be less than 1.0, as the proxy can drop, delay, or modify packets to evade the detection algorithm. However, it is also important to observe whether imperfect forwarding conditions lead to a graceful degradation or a sharp decrease in the algorithm's performance. The principles outlined above represent independent variables in the live analysis experiments, and thus we give the same consideration to these variables as those in Section 4.3.1. We assume that imperfect conditions are mutually exclusive, so each experiment changes one independent variable while assuming perfect conditions for the other variables.

We base the values for the independent variables on the values for the timestamp and packet size tolerance values from the detection algorithm, as we expect the algorithm performance to decrease if one or both of the tolerance values are exceeded. For perfect conditions, all independent variable values are set to 0. Since we set T to 0.1 seconds and S to 20 bytes in experiments, we give values of 0.08 and 0.0825 seconds to the delay variable, 22 and 25 bytes to the packet size delta variable, and 0.1 and 0.2 to the drop probability variable. We further elaborate on the implications of these choices in the Results chapter.

Chapter 5

Results

This chapter presents results for the remote access and proxy detection experiments described in the Evaluation chapter and discusses inferences we make from the results.

5.1 Remote Access Results

Client-Side Scheme	**Port Knocking**	**ENR**	**DNS Lookup**
Average Computation Delay (ms)	120.567	67.133	49.468
Standard Deviation (ms)	30.976	20.564	2.601

Table 5.1: Delay measurements for client-side remote access computations

Following the experiments outlined in Section 4.2, we obtain delay measurements on the client and router endpoints for each remote access scheme. Table 5.1 summarizes the client-side results with an average measurement and standard deviation over thirty trials. First, we observe that port knocking takes roughly twice as long as ENR to complete; this difference stems from the direct variation between the delay measurement and the number of ports in the stream that the client sends in the port knocking approach. The ENR approach only requires one packet that contains the authentication request and maintains a relatively constant delay, whereas port knocking sends a number of packets equal to the number of ports in the stream. Reducing the number of ports would decrease the delay measurement in a roughly linear fashion; however, we note that a smaller port stream would make it easier for an on-path adversary to sample the correct port sequence. Nevertheless, given that the client authenticates as needed, a user requesting remote access is unlikely to notice the delay measurements presented in Table 5.1. Second, we observe that the average DNS lookup time is within the accepted range of 20-120 milliseconds [42], and we do not anticipate that the DNS server's placement would significantly impact this value. As described in Section 4.2.3, we add the DNS lookup time to each of the delays that port knocking and ENR incur, but this addition is small enough to be efficient for client use.

Table 5.2 summarizes the router-side delay measurements for port knocking and ENR, as there is no unique router module for DNS. Section 4.2.1 mentions that we compute the maximum and average delay measurement because the router module acts differently based on the properties of each incoming packet. For port knocking, the maximum is several standard deviations outside the average and the standard deviation is larger than the average. These properties suggest that router-side delay skews to the right. This observation

Router-Side Scheme	**Port Knocking**	**ENR**
Maximum Computation Delay (ms)	2.96	3.223
Average Computation Delay (ms)	0.0384	2.635
Standard Deviation (ms)	0.125	0.530

Table 5.2: Delay measurements for router-side remote access computations

is consistent with port knocking's behavior because the router module ignores the majority of incoming requests, as the port number is unlikely to match or complete a valid sequence. The maximum value occurs when the router module creates an access rule in response to a completed port sequence, whereas the largely ignored packets contribute to the low average. ENR's distribution does not skew as much, but its average computation delay is much higher because the router must respond to every authentication request that it receives, even if the router did not create an access rule for that request. For ENR, the router only processes packets that arrive on a specific port, rather than listening for traffic arriving on all ports. Thus, the higher average value is acceptable given the amount of traffic that ENR processes compared to port knocking. Delay measurements for both schemes are on the order of milliseconds and could be run on a router without drastic performance impacts.

Overall, port knocking and ENR are viable options for remote authentication, and leveraging DNS provides a lightweight mechanism to rotate access tokens as needed. Both approaches have their advantages and disadvantages from the perspective of home router security and user privacy. As an example, Section 2.1 discusses the potential for IoT devices to expose sensitive user data in network communications. The ramifications of discovering port sequences, perhaps as TXT records on a DNS server or as files on the home router, could allow anyone with the compromised information to access an internal service. With ENR, the client ID and MAC key for that client would have to be exposed together to be useful to an adversary. Additionally, because accesses are tied to that client, it is easier to audit for unauthorized behavior and quarantine the misbehaving party. As another example, port knocking exhibits a lower level of visibility than ENR because the router does not respond to incoming traffic. Thus, an on-path adversary cannot infer as much from the traffic that port knocking generates. In fact, because the port knocking approach in this paper resembles a traditional port scan, the adversary may disregard the observed traffic as an automated scan. The router response in ENR indicates that bidirectional communication takes place and may prompt more interest from an adversary than port knocking. Homeowners could select either mechanism, with or without supporting DNS infrastructure, to best suit the properties of their home networks.

5.2 Proxy Detection Results

Section 4.3 describes the experimental setups to evaluate the proxy detection algorithm and defines the independent variables that influence the algorithm's performance. We present runtime results first, followed by results for the live analysis experiments.

5.2.1 Algorithm Runtime

The proxy detection algorithm's runtime results are illustrated in Figures 5.1 and 5.2. We note that the standard deviation values are not large enough to be displayed on the runtime charts, so we omit them here. First, we observe that the runtime output matches what we expect, since the best-case runtime is at least quadratic and the average runtime resembles a cubic or quartic curve. In both experiments, the curve becomes increasingly steep as we vary the secondary independent variable, but the algorithm's asymptotic time complexity remains the same.

Second, we observe that the runtime output is relatively similar across both charts, and thus one computation does not inherently dominate the other. From Section 4.3.2, we mention that the first runtime experiment assumes that a proxy device handles many small connection requests, and the second experiment assumes that a proxy device handles a few large connection requests. Based on the runtime output, the algorithm works equally well for both workloads. We note that if a home network contains a proxy that handles many large connection requests in a short period of time, the detection algorithm could stop temporarily or switch to a less involved computation. We elaborate on this possibility in Section 6.1.2.

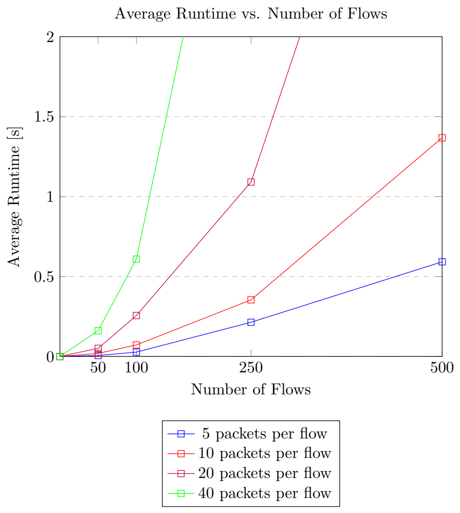


Figure 5.1: Runtime measurements with respect to the number of unique flows

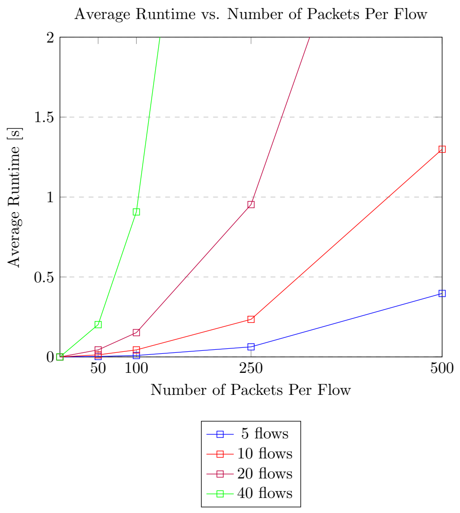


Figure 5.2: Runtime measurements with respect to the number of packets per flow

With respect to the runtime values themselves, we observe that the algorithm performs fast enough for use in home networks. Although some of the average runtime values are much greater than 2 seconds, the experiment configurations that produce these values are unlikely to be achieved in practice. First, Section 3.4 introduces two modes of operation for the proxy detection algorithm: the live mode calculates scores for flows observed on a live network, and the offline mode calculates scores for flows from a packet capture collected in the past. The significance of the algorithm runtime for the live mode relies on the window size. A smaller window length helps limit the number of unique flows observed in each window and the number of packets observed per flow; thus, the algorithm runtime with a small window would fall somewhere on the lower left portion of both runtime charts. However, the window length should be large enough to observe a sufficient amount of traffic, as a high score is relatively meaningless if only a few packets are compared between each flow pair. Overall, a home router could regularly perform the algorithm's computation without significantly interfering with its other responsibilities.

Variable Values	**All 0**	**Delay = 0.08**	**Delay = 0.0825**	**Size Delta = 22**	**Size Delta = 25**	**Drop Rate = 0.1**	**Drop Rate = 0.2**
Average Score	0.985	0.893	0.827	0.923	0.802	0.893	0.825
Standard Deviation	0.02	0.058	0.047	0.04	0.075	0.043	0.064

Table 5.3: Score measurements for the TLS tunneling experiments

Variable Values	**All 0**	**Delay = 0.08**	**Delay = 0.0825**	**Size Delta = 22**	**Size Delta = 25**	**Drop Rate = 0.1**	**Drop Rate = 0.2**
Average Score	0.975	0.871	0.839	0.886	0.811	0.876	0.791
Standard Deviation	0.02	0.066	0.07	0.042	0.061	0.058	0.058

Table 5.4: Score measurements for the HTTPS request experiments

5.2.2 Live Analysis

The detection algorithm's average score and standard deviation for the flow pairs of interest are detailed in Table 5.3 for the TLS tunneling experimental setup and in Table 5.4 for the HTTPS request experimental setup. Each column represents a unique configuration of independent variables, including the delay in seconds before forwarding a packet, the number of bytes to add or remove from a packet, and the probability that a packet is dropped. For perfect forwarding conditions in which all variables are set to 0, the average scores are roughly 1.0 as expected. The reported value is likely because the router misses packets that pass through the network while the detection algorithm runs.

Before explaining the results for imperfect forwarding conditions, it is important to revisit the choices for the independent variable values. Section 3.4 mentioned that the algorithm calculates the final score for a flow pair as a ratio; the numerator represents the number of matched packets between the incoming and outgoing flows, and the denominator represents the number of possible matches. Since the send rate is set to a constant value in experiments, the denominator's value remains roughly the same in each window. The numerator, however, is expected to decrease proportionally with the values of the independent variables. The drop rate values yield the simplest explanation: if the proxy drops approximately X percent of packets before sending them to the destination, the router does not record these packets for the outgoing flow. Therefore, the percentage of matches and the flow pair score decrease by the value of X. The experimental outputs with respect to varying the drop rate generally support this observation.

While not as obvious as the drop rate, the choices for the delay and size delta also follow this trend. Given a packet size tolerance of 20 bytes and a size delta of 22, a packet passing through the proxy has its total packet size changed by a value in the range of -22 to 22 bytes. The fraction of size changes that push the packet's size outside the tolerance value is 4/45, a value close to 0.1 under the assumption that packet sizes are restricted to integer values. Using a size delta of 25 gives a fraction of about 0.2; therefore, we expect that varying the size delta with the selected values produces similar scores to varying the drop rate. The experimental outputs support the expected behavior and reinforce the observation that decreasing the likelihood of a match decreases the resulting score by roughly the same amount. The values for delay are more difficult to calibrate, as the elapsed time for the proxy to repackage and send the message towards the destination must be factored into any manually-added delay. Given a timestamp tolerance of 0.1 seconds, the delay values listed in Table 5.3 decrease the number of matches and observed score by approximately 10 and 20 percent, respectively.

It should be noted that the timestamp and packet size indicators for proxy behavior are sensitive to a variety of imperfect forwarding conditions beyond those we examine. For example, unintentional delays from local network congestion or a lagging proxy device can result in the timestamp tolerance value being exceeded for several packets in a flow pair. In fact, an attacker with control of a proxy device could intentionally delay communication to evade the detection scheme. With respect to packet sizes, an attacker with traffic interception capabilities on the proxy device possesses the freedom to add, remove, or modify packet headers or payloads. Certainly, one limitation of the proxy detection scheme is its reliance on a set of assumptions about how proxy devices are expected to behave; the experimental setups presented in this paper follow the same set of assumptions and thus yield fairly strong performance results. We discuss this limitation further in the Discussion chapter.

Chapter 6

Discussion

The Results chapter provides supporting evidence for the feasibility of remote access and proxy detection schemes in home networks. In this chapter, we focus on limitations of our work, as well as extensions to our work that address these limitations. Finally, we offer concluding remarks.

6.1 Limitations and Future Work

While the possibilities for home network defense models are unbounded, we focus on extensions to the services presented in this paper. One path for future work examines how remote access and proxy detection would function together. The other path addresses a more efficient way to analyze network traffic for the proxy detection algorithm.

6.1.1 Combining Security Services

Our experiments consider two classes of security services independently. The remote access experiments operate without regard for internal traffic inspection, and the proxy detection experiments assume that external communication requests are permitted without prior authentication. One limitation of separating the services is that neither approach simultaneously deters attacks outside and inside the network. Instead, each service focuses on one attacker type. However, it is possible to create a setup where the two service classes coexist; Section 2.3.2 discusses the potential for user-controlled proxies to offer more powerful access capabilities to the homeowner. The multilayered defense model alerts homeowners to remote-controlled proxies, increases the difficulty of launching a successful network intrusion, and allows legitimate proxies to exist inside the network.

Section 3.4 notes that the proxy detection algorithm does not make any decisions on its own; rather, the router can act on the scores provided by the algorithm to block flows from a source or quarantine a device completely. Combining the detection algorithm with remote access would require a small change to the router's behavior; namely, the router would check any access rules created from successful authentication requests before acting in response to a high score. In the hybrid experimental setup, the client would first authenticate to gain access to the proxy device using port knocking or the explicit network request protocol. Then, the client would use the proxy device to tunnel TLS traffic or issue HTTPS requests. For this scenario, the router is not expected to block any flows due to the access rule that was created for the remote user.

The original setups for proxy detection would result in blocking, because the client assumed control of the proxy device without issuing an authentication request.

6.1.2 Using SDN Statistics for Proxy Detection

The live analysis mode for residential proxy detection requires the router to process every ingoing and outgoing packet. Although the runtime results showed that the algorithm itself produces reasonable overhead, it may be possible to reduce the number of required comparisons by filtering out flows that are unlikely to be useful. In the live analysis experiments, most flow pairs other than the flow pair of interest produced scores of zero or close to zero. It is unlikely that the majority of flow pairs will signal proxying behavior; therefore, the proxy detection mechanism could benefit significantly from minimizing the amount of time that the router observes traffic on a per-packet basis.

Section 2.3.4 discusses potential benefits of software-defined networking in home networks, such as implementing custom traffic analysis. It is possible to augment the OpenWrt distribution on the TP-Link router with OpenFlow, a network protocol that makes programming with SDN possible, and with OpenvSwitch, an implementation of a multilayer network switch. Other works have leveraged these tools to detect network-based attacks based on high-level flow statistics. Giotis et al. [43] use the native OpenFlow specification to detect Denial of Service attacks by observing statistics such as the amount of upload traffic on a given port. Nobakht et al. [44] propose an intrusion detection mechanism that monitors network activity of IoT devices and matches this activity against known patterns of attack traffic. Both works claim that analyzing all traffic at the deepest level is not efficient or scalable, regardless of the attack vector being examined. Rather, they observe flow statistics as aggregate quantities and analyze individual flows as needed. An aggregate approach to detecting proxy behavior could also address the limitation brought up in Section 5.2.2 about the sensitivity of per-packet indicators. Delaying forwarding intentionally or applying large packet size changes dampens the effectiveness of the detection algorithm, so a weaker filtering mechanism could catch behavior that is undetectable at a more granular level.

The live analysis mode for proxy detection would be reshaped as a two-step process. In the first step, an SDN controller would continuously gather flow statistics using the native versions of OpenFlow and OpenvSwitch; the OpenFlow protocol supports OFPMP_AGGREGATE, a message type that requests aggregate information for multiple flow entries. Like the original module, the router would periodically analyze the aggregate data and look for any signs of proxying behavior. The granularity of the aggregate should be considered carefully, as collecting flow data from one source address will provide a different result compared to flow data from all source addresses. Additionally, the indicators for proxy behavior would not be as strict in this stage because aggregate flow data is not as expressive as individual packet data. If the router decides that there is enough evidence to inspect traffic more closely, it will move to the second step. The OpenFlow protocol also supports OFPMP_FLOW, a message type that requests information about individual flow entries. The second step is largely the same as the original approach; flow pairs are scored based on the forwarding indicators described in Section 3.4. The router could run the detection algorithm itself or elevate packets to an SDN controller in the cloud. This process would only last for a short time, perhaps for a few windows, to give the router enough information to make a decision. Then, traffic analysis would return to aggregate statistics gathering. The cyclic nature of this mechanism allows for more efficient and scalable traffic analysis that makes proxy detection more feasible for home networks.

6.2 Conclusion

In this paper, we present security services that defend a home network from intrusions and misuse of home-networked devices. Remote access mechanisms increase the barrier to entry and limit the attacker's ability to perform reconnaissance and steal sensitive user data. Cyber deception techniques add ambiguity to the value of home network resources and decrease the perceived benefits of controlling these resources. Finally, residential proxy detection increases an attacker's risk of being blocked from the network and of being noticed on the network, perhaps to the benefit of law enforcement. Together, these services diminish the value of the target on home networks, potentially causing attackers to ignore home networks entirely. We have shown that these services can run on a consumer-grade router with minimal overhead while performing effectively.

For every attack model that puts residential networks at risk, there may be several ways to defend against the attack. Our work aims not only to bring attention to the security needs of home networks, but also to highlight the capabilities of home routers to act as network defenders. The flexibility of open-source software allows many types of security services to be deployed, tested and improved transparently. Although poor security practices may continue to persist in home networks, these services would negate the harmful effects of some practices and protect homeowners without sufficient technical expertise. Above all, the goal of improving residential network security is to give homeowners peace of mind when using technologies that are intended to benefit them.

Bibliography

[1] SonicWall, "New sonicwall research finds aggressive growth in ransomware, rise in iot attacks." [Online]. Available: https://www.sonicwall.com/news/new-sonicwall-research-finds-aggressive-growth-in-ransomware-rise-in-iot-attacks/ Accessed:April 7, 2021.

[2] A. Spadafora, "Mirai botnet returns to target iot devices." [Online]. Available: https://www.techradar.com/news/mirai-botnet-returns-to-target-iot-devices Accessed:April 7, 2021.

[3] Cybersecurity & Infrastructure Security Agency, "Security tip (st15-002); home network security." [Online]. Available: https://us-cert.cisa.gov/ncas/tips/ST15-002 Accessed:February 21, 2021.

[4] P. Weidenbach and J. vom Dorp, "Home router security report 2020," June 2020.

[5] M. Powell, "Wi-fi router security knowledge gap putting devices and private data at risk in uk homes." [Online]. Available: https://www.broadbandgenie.co.uk/blog/20180409-wifi-router-security-survey Accessed:February 21, 2021.

[6] D. Kumar, K. Shen, B. Case, D. Garg, G. Alperovich, D. Kuznetsov, R. Gupta, and Z. Durumeric, "All things considered: an analysis of iot devices on home networks," in *28th {USENIX} Security Symposium ({USENIX} Security 19)*, 2019, pp. 1169–1185.

[7] J. Valente, M. A. Wynn, and A. A. Cardenas, "Stealing, spying, and abusing: Consequences of attacks on internet of things devices," *IEEE Security & Privacy*, vol. 17, no. 5, pp. 10–21, 2019.

[8] iTel Networks, "The difference between symmetrical & asymmetrical connections." [Online]. Available: https://itel.com/what-is-the-difference-between-symmetrical-and-asymmetrical-connections/ Accessed:February 21, 2021.

[9] Nmap, "Scan time reduction techniques." [Online]. Available: https://nmap.org/book/reduce-scantime.html Accessed:February 21, 2021.

[10] X. Mi, S. Tang, Z. Li, X. Liao, F. Qian, and X. Wang, "Your phone is my proxy: Detecting and understanding mobile proxy networks," *Network and Distributed Systems Security (NDSS) Symposium*, 2021.

[11] X. Mi, X. Feng, X. Liao, B. Liu, X. Wang, F. Qian, Z. Li, S. Alrwais, L. Sun, and Y. Liu, "Resident evil: Understanding residential ip proxy as a dark service," in *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2019, pp. 1185–1201.

[12] A. Hanzawa and H. Kikuchi, "Analysis on malicious residential hosts activities exploited by residential ip proxy services," in *International Conference on Information Security Applications*. Springer, 2020, pp. 349–361.

[13] J. Choi, M. Abuhamad, A. Abusnaina, A. Anwar, S. Alshamrani, J. Park, D. Nyang, and D. Mohaisen, "Understanding the proxy ecosystem: A comparative analysis of residential and open proxies on the internet," *IEEE Access*, vol. 8, pp. 111 368–111 380, 2020.

[14] R. Wright, "Fbi: How we stopped the mirai botnet attacks." [Online]. Available: https://searchsecurity.techtarget.com/news/252459016/FBI-How-we-stopped-the-Mirai-botnet-attacks Accessed:February 21, 2021.

[15] Tor, "The tor project: Anonymity online." [Online]. Available: https://www.torproject.org/ Accessed:March 17, 2021.

[16] D. Isabel, "Port knocking: Beyond the basics," *GIAC Security Essentials Certification (GSEC) Practical Assignment Version*, vol. 1, 2005.

[17] R. Degraaf, J. Aycock, and M. Jacobson, "Improved port knocking with strong authentication," in *21st Annual Computer Security Applications Conference (ACSAC'05)*. IEEE, 2005, pp. 10–pp.

[18] A. Haber, J. G. R. De Mier, and F. Reichert, "Virtualization of remote devices and services in residential networks," in *2009 Third International Conference on Next Generation Mobile Applications, Services and Technologies*. IEEE, 2009, pp. 182–186.

[19] B. Parno, D. Wendlandt, E. Shi, A. Perrig, B. Maggs, and Y.-C. Hu, "Portcullis: Protecting connection setup from denial-of-capability attacks," *ACM SIGCOMM Computer Communication Review*, vol. 37, no. 4, pp. 289–300, 2007.

[20] C. A. Shue, A. J. Kalafut, M. Allman, and C. R. Taylor, "On building inexpensive network capabilities," *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 2, pp. 72–79, 2012.

[21] C. R. Taylor, D. C. MacFarland, D. R. Smestad, and C. A. Shue, "Contextual, flow-based access control with scalable host-based sdn techniques," in *IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications*. IEEE, 2016, pp. 1–9.

[22] A. Tosun, M. De Donno, N. Dragoni, and X. Fafoutis, "Resip host detection: Identification of malicious residential ip proxy flows," in *39th IEEE International Conference on Consumer Electronics*. IEEE, 2021.

[23] M. Antonakakis, R. Perdisci, D. Dagon, W. Lee, and N. Feamster, "Building a dynamic reputation system for dns." in *USENIX security symposium*, 2010, pp. 273–290.

[24] E. Stalmans and B. Irwin, "A framework for dns based detection and mitigation of malware infections on a network," in *2011 Information Security for South Africa*. IEEE, 2011, pp. 1–8.

[25] P. Carvalho, B. Dantas, S. R. Lima, and J. Marco, "Detection of anonymised traffic: Tor as case study," *ruSMART 2020*, 2020.

[26] Nginx, "Nginx." [Online]. Available: https://nginx.org/en/ Accessed:March 21, 2021.

[27] X. Han, N. Kheir, and D. Balzarotti, "Deception techniques in computer security: A research perspective," *ACM Computing Surveys (CSUR)*, vol. 51, no. 4, pp. 1–36, 2018.

[28] W. Han, Z. Zhao, A. Doupé, and G.-J. Ahn, "Honeymix: Toward sdn-based intelligent honeynet," in *Proceedings of the 2016 ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization*, 2016, pp. 1–6.

[29] C.-Y. J. Chiang, S. Venkatesan, S. Sugrim, J. A. Youzwak, R. Chadha, E. I. Colbert, H. Cam, and M. Albanese, "On defensive cyber deception: A case study using sdn," in *MILCOM 2018-2018 IEEE Military Communications Conference (MILCOM)*. IEEE, 2018, pp. 110–115.

[30] S. Sugrim, S. Venkatesan, J. A. Youzwak, C.-Y. J. Chiang, R. Chadha, M. Albanese, and H. Cam, "Measuring the effectiveness of network deception," in *2018 IEEE International Conference on Intelligence and Security Informatics (ISI)*. IEEE, 2018, pp. 142–147.

[31] S. Sundaresan, S. Burnett, N. Feamster, and W. De Donato, "Bismark: A testbed for deploying measurements and applications in broadband access networks," in *2014 {USENIX} Annual Technical Conference ({USENIX}{ATC} 14)*, 2014, pp. 383–394.

[32] A. K. Simpson, F. Roesner, and T. Kohno, "Securing vulnerable home iot devices with an in-hub security manager," in *2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*. IEEE, 2017, pp. 551–556.

[33] C. R. Taylor, T. Guo, C. A. Shue, and M. E. Najd, "On the feasibility of cloud-based sdn controllers for residential networks," in *2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*. IEEE, 2017, pp. 1–6.

[34] C. R. Taylor and C. A. Shue, "Validating security protocols with cloud-based middleboxes," in *2016 IEEE Conference on Communications and Network Security (CNS)*. IEEE, 2016, pp. 261–269.

[35] Y. Liu, C. R. Taylor, and C. A. Shue, "Authenticating endpoints and vetting connections in residential networks," in *2019 International Conference on Computing, Networking and Communications (ICNC)*. IEEE, 2019, pp. 136–140.

[36] Y. Liu and C. A. Shue, "Community cleanup: Incentivizing network hygiene via distributed attack reporting," in *NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 2020, pp. 1–9.

[37] T. Hunter, P. Terry, and A. Judge, "Distributed tarpitting: Impeding spam across multiple servers." in *LISA*, vol. 3, 2003, pp. 223–236.

[38] V. Sivaraman, H. H. Gharakheili, A. Vishwanath, R. Boreli, and O. Mehani, "Network-level security and privacy control for smart-home iot devices," in *2015 IEEE 11th International conference on wireless and mobile computing, networking and communications (WiMob)*. IEEE, 2015, pp. 163–167.

[39] D. C. MacFarland and C. A. Shue, "The sdn shuffle: creating a moving-target defense using host-based software-defined networking," in *Proceedings of the Second ACM Workshop on Moving Target Defense*, 2015, pp. 37–41.

[40] OpenWrt, “Welcome to the openwrt project.” [Online]. Available: https://openwrt.org/ Accessed:March 9, 2021.

[41] J. Ellingwood, “How to configure bind as an authoritative-only dns server on ubuntu 14.04.” [Online]. Available: https://www.digitalocean.com/community/tutorials/how-to-configure-bind-as-an-authoritative-only-dns-server-on-ubuntu-14-04 Accessed:March 9, 2021.

[42] GTMetrix, “Yslow: Reduce dns lookups.” [Online]. Available: https://gtmetrix.com/reduce-dns-lookups.html Accessed:March 28, 2021.

[43] K. Giotis, C. Argyropoulos, G. Androulidakis, D. Kalogeras, and V. Maglaris, “Combining openflow and sflow for an effective and scalable anomaly detection and mitigation mechanism on sdn environments,” *Computer Networks*, vol. 62, pp. 122–136, 2014.

[44] M. Nobakht, V. Sivaraman, and R. Boreli, “A host-based intrusion detection and mitigation framework for smart home iot using openflow,” in *2016 11th International conference on availability, reliability and security (ARES)*. IEEE, 2016, pp. 147–156.