IoTLS: Understanding TLS Usage in Consumer IoT Devices

Muhammad Talha Paracha
Northeastern University

Daniel J. Dubois
Northeastern University

Narseo Vallina-Rodriguez
IMDEA Networks / ICSI / AppCensus Inc.

David Choffnes
Northeastern University

ABSTRACT

Consumer IoT devices are becoming increasingly popular, with most leveraging TLS to provide connection security. In this work, we study a large number of TLS-enabled consumer IoT devices to shed light on how effectively they use TLS, in terms of establishing secure connections and correctly validating certificates, and how observed behavior changes over time. To this end, we gather more than two years of TLS network traffic from IoT devices, conduct active probing to test for vulnerabilities, and develop a novel blackbox technique for exploring the trusted root stores in IoT devices by exploiting a side-channel through TLS *Alert Messages*. We find a wide range of behaviors across devices, with some adopting best security practices but most being vulnerable in one or more of the following ways: use of old/insecure protocol versions and/or ciphersuites, lack of certificate validation, and poor maintenance of root stores. Specifically, we find that *at least* 8 IoT devices still include distrusted certificates in their root stores, 11/32 devices are vulnerable to TLS interception attacks, and that many devices fail to adopt modern protocol features over time. Our findings motivate the need for IoT manufacturers to audit, upgrade, and maintain their devices' TLS implementations in a consistent and uniform way that safeguards all of their network traffic.

CCS CONCEPTS

• **Security and privacy** → **Network security**; **Embedded systems security**; • **Networks** → *Network measurement*; **Network security**;

KEYWORDS

Internet of Things, IoT, Transport Layer Security, TLS, network security, embedded systems security, measurement techniques

ACM Reference Format:
Muhammad Talha Paracha, Daniel J. Dubois, Narseo Vallina-Rodriguez, and David Choffnes. 2021. IoTLS: Understanding TLS Usage in Consumer IoT Devices. In *ACM Internet Measurement Conference (IMC '21), November 2–4, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3487552.3487830

IMC '21, November 2–4, 2021, Virtual Event, USA

ACM ISBN 978-1-4503-9129-0/21/11…$15.00
https://doi.org/10.1145/3487552.3487830

1 INTRODUCTION

Consumer Internet-of-Things (IoT) devices such as voice assistants, smart TVs and video doorbells are popular, with their prevalence projected to be 75 billion by 2025 [14]. Most IoT devices rely on TLS, the de facto secure transport protocol, to provide confidentiality, integrity and authenticity of their network communications [26]. Numerous prior works have shown that TLS security properties can be compromised due to development errors (e.g., [31]), insecure configurations (e.g., [39]), and outdated clients (e.g., [20]). While TLS usage has been studied extensively in mobile applications and web browsers (e.g., [47], [49], [37]), there is little insight into its effectiveness in the IoT ecosystem (e.g., [26]).

More specifically, there exists a research gap in understanding whether TLS implementations in IoT devices: (*i*) establish connections using secure TLS versions and ciphersuites, (*ii*) correctly perform certificate validation while using a generally trusted set of root certificates, and (*iii*) adopt new features as the protocol evolves over time (e.g., modern ciphersuites). There are several challenges that prevent the use of existing methodologies to study these aspects of IoT devices. *First*, understanding TLS support on a significant number of IoT devices requires blackbox testing techniques; this is because source code is generally unavailable and firmware analysis is not scalable. *Second*, most IoT devices provide limited ways to trigger TLS traffic for measurement—the timing, destination, and contents of their communication are all dependent on device functionality and interactions. *Third*, existing vantage points offer limited opportunities to track device behavior over time (e.g., recent work considers only manufacturer-level device tracking using ISP/IXP data [53]).

In this work, we address these challenges to study a large number of TLS-enabled consumer IoT devices (with over 200 million units sold collectively). *First*, we shed light on the security of TLS implementations and configurations in these devices using existing and novel active measurement techniques that require only TLS traffic interception. *Second*, based on the insight that devices generate significant network traffic when powered on, we automate device reboots using smart plugs to trigger TLS activity for our experiments. And *third*, we analyze ≈2 years of network traffic from these devices via uncontrolled experiments to study how their TLS usage changes over time. Altogether, we conduct *active* experiments on 32 devices, and collect *passive* data from 40 devices, each generating TLS traffic for at least 6 months.

Our main goals are to evaluate the security of TLS connections established by IoT devices, how this changes over time, and whether they correctly validate certificates. More specifically, we study how devices' TLS implementations are configured with respect to TLS versions and ciphersuites supported, and provide the first longitudinal analysis of how these properties change over time, as the TLS

protocol and attacks against it evolve. We further check whether the devices properly validate certificates to protect against the TLS interception attacks, extending prior work by including a more comprehensive set of invalid certificates in our tests. We develop a novel active testing strategy to reveal the trusted set of root certificates on a device. These certificates form the *"trusted root"* of all security guarantees provided by the TLS protocol and auditing them is particularly important given the recent rise in supply-chain attacks by powerful adversaries [23, 54]. While prior works have studied root stores in open platforms (e.g., operating systems and browsers), to the best of our knowledge we are the first to investigate the validity of root certificates used in IoT devices.

Our main research findings are listed below:

- The vast majority of devices establish connections using TLS 1.2 with secure ciphersuites, but rarely adopt TLS 1.3 or stop advertising insecure ciphersuites (DES, 3DES, RC4, EXPORT) over time. Surprisingly, devices that *do* support the latest TLS versions and strongest ciphersuites often encounter lack of server support. In addition, 7 devices downgrade to deprecated protocol versions or old ciphersuites in the face of an active on-path attacker.
- 11 devices are vulnerable to TLS interception attacks—they either bypass certificate validation altogether or do not validate hostnames. Moreover, TLS connections from 7 vulnerable devices contained sensitive data that can be exposed to attackers. We responsibly disclosed these vulnerabilities to device vendors, and at the time of writing one vendor confirmed that the issue was fixed in response to our disclosure.
- *At least* 8 devices contain unexpired-yet-deprecated root certificates in their trusted root stores. Moreover, all of these devices trusted *at least* one CA that has been explicitly distrusted by popular browsers due to misbehavior (e.g, *WoSign, TurkTrust, Certinomis, CNNIC*). In one device, CAs that were deprecated as early as 2013 were present.
- TLS fingerprints from these devices reveal that multiple devices likely use the same (often vulnerable) TLS library, and individual devices likely include multiple TLS libraries. For the former, shared libraries mean that attackers can use knowledge of the fingerprints and associated vulnerabilities to scale their attacks to large numbers of devices. For the latter, multiple TLS libraries hint at more opportunities for supply chain attacks (if different implementations come from different vendors) and make TLS security more challenging to maintain.

To ensure reproducibility and enable new research, we have made all of our longitudinal TLS handshake data, controlled experimentation data and analysis software publicly available at: https://github.com/NEU-SNS/IoTLS.

2 BACKGROUND

Transport Layer Security (TLS) is the de facto, IETF-standard, Internet security protocol to provide confidentiality, integrity, and authenticity of network communications. Since Netscape started work on the TLS predecessor, Secure Sockets Layer (SSL), ≈25 years ago, the protocol has undergone rigorous development featuring various standardization efforts and releases—SSL 2.0 (1995), SSL 3.0 (1996), TLS 1.0 (1999), TLS 1.1 (2006), TLS 1.2 (2008) and TLS 1.3 (2018). This section provides relevant background information about TLS.

Root Stores TLS generally uses digital certificates that bind host identities with cryptographic material. These certificates are issued by Certificate Authorities (CAs) and can be "revoked" if they get compromised. Server authentication is the most common TLS usage where clients store relevant information about one or more trusted CAs root certificates and require that servers present valid certificates from one of them to authenticate themselves. These certificates form a trusted set of "root" store certificates deployed on end systems; if the private key for any of these trusted root certificates is compromised, the attacker can circumvent security guarantees of *all* TLS connections by a client.[1] Currently, web browsers and operating systems ship with dozens of root certificates in their root stores to enable TLS communication with a wide range of servers. Due to the crucial importance of root certificates, these platforms actively maintain their root stores to remove any certificates from CAs that violate CA guidelines or get compromised.

Secure connection establishment A TLS "handshake" is the set of messages that establishes a secure session between two end hosts. The process is initiated by a client to advertise its supported protocol versions, ciphersuites (i.e., cryptographic algorithms), and extensions (i.e., other advanced features). In response, a TLS server decides the protocol version and ciphersuite to use based on its compatibility. During the handshake, the client and server can authenticate each other and compute cryptographic keys to be used for confidentiality and integrity of the future communication. The authentication typically involves validating the received certificate chain. Any data sent after a successful handshake is encrypted.[2] TLS *Alert Messages* can be sent at any point to notify the other party of any errors (e.g., incompatible protocol version, signature verification failure during certificate validation) or to simply close the connection.

A TLS *ClientHello* can be used to infer the software responsible for generating that connection. That is because clients and libraries vary in the protocol versions, ciphersuites, extensions, and other features they support. A TLS "fingerprint" is a permutation of these features obtained from *ClientHello*s of a known application to enable its detection in passively monitored TLS traffic from unknown sources [38, 54]. We define a *TLS instance* as TLS implementation (e.g., software library) and configuration (e.g., selected ciphersuites and extensions) that collectively produce a given TLS fingerprint.

Ciphersuites Ciphersuites have also evolved over time. Deprecated ones need to be avoided for secure connection establishment. More specifically, any usage of ciphersuites involving (DES, 3DES, RC4 and EXPORT) demands for "immediate" remediation [5, 21] because of their vulnerability to attacks including, but not limited to, Biased Keystreams (2013) [25], FREAK (2015) [28], Logjam (2015) [24], and Sweet32 (2016) [29]. Further, *ClientHellos* using (NULL or ANON) ciphersuites do not offer authentication or encryption and, as such, can only be used for specific (insecure) use cases. Finally, modern ciphersuites involving (DHE or ECDHE) offer perfect forward secrecy (i.e., protection of data communicated in

[1] Except for applications that use strategies such as key pinning.
[2] Assuming the NULL ciphersuite is not selected.

Table 1: List of the 40 TLS-supporting devices in our study. (*) denotes devices used only in *passive* experiments.

Cameras (n = 7)	Smart Hubs (n = 7)	Home Automation (n = 7)	TV (n = 5)	Audio (n = 7)	Appliances (n = 7)
Blink Camera*	Blink Hub	Smartlife Bulb	Fire TV	Google Home Mini	GE Microwave
Amazon Cloudcam*	Smartthings Hub	Smartlife Remote	Samsung TV*	Amazon Echo Plus	Samsung Washer*
Zmodo Doorbell	Philips Hub	Meross Dooropener	LG TV	Amazon Echo Dot	Samsung Dryer
Yi Camera	Wink Hub 2	TP-Link Bulb	Roku TV	Amazon Echo Dot 3	Samsung Fridge
D-Link Camera	Sengled Hub*	Nest Thermostat	Apple TV	Amazon Echo Spot	Smarter iKettle
Amcrest Camera	Switchbot Hub	TP-Link Plug		Harman Invoke	Behmor Brewer
Ring Doorbell*	Insteon Hub*	Wemo Plug		Apple HomePod	LG Dishwasher*

the past despite a future compromise of secret keys) and therefore should be adopted.

Vulnerabilities and attacks By 2020, major browsers had deprecated all TLS versions below 1.2 due to serious security flaws [3]. Yet, some TLS clients voluntarily downgrade connection security upon handshake failure to improve compatibility with older servers. The POODLE (2014) [46] attack exploited the behavior of major web browsers and other clients to fallback to SSL 3.0 (known to be insecure) and highlighted the risks of any fallback behavior. On the other hand, the TLS interception class of attacks typically refer to weaknesses in handshake validation that are exploited by on-path attackers. These attacks are particularly severe as they allow secret eavesdropping of all TLS communication sent between a client and server on a compromised connection.

3 GOALS & ASSUMPTIONS

Our goals are to answer three research questions (RQs):

RQ1: Do devices *securely establish* TLS connections? Securely establishing TLS connections means that devices use secure TLS versions and ciphersuites. In this paper, we consider whether devices are resilient to in-network adversaries (e.g., a network provider) that have the ability to capture and manipulate TLS traffic between an IoT device and the destinations it contacts. We focus on device support for the latest, secure protocol versions, modern ciphersuites, and negotiated TLS configurations between IoT clients and their destinations. We use ≈2 years of passively collected longitudinal IoT traffic to determine whether devices adopt new features and abandon deprecated ones.

RQ2: Do devices *properly validate* TLS certificates? In this paper, we focus on evaluating whether devices accept connections with invalid certificates, and understanding whether their root stores contain deprecated and/or distrusted root certificates. Specifically, we focus on validation of the server certificate chain, hostname and various X.509 extensions specified in RFC 2818 [18] and RFC 5280 [19]. In addition, we evaluate the configured set of trusted root certificates to determine whether devices protect against distrusted and/or stale root certificates.

RQ3: What is the *diversity of behaviors* within and across devices? The effectiveness of a single attack vector is limited to the set of devices that share the same vulnerability. To understand the breadth of the impact of potential attacks, we investigate how many devices exhibit the same TLS behavior and, potentially, the same security issues. We further investigate whether individual devices exhibit different TLS behavior for different connections—indicative of multiple TLS instances on the same device.

To answer these questions, instead of modeling IoT devices as monolithic implementations, we treat them as complex devices that can integrate third-party components and even allow users to install third-party software as in the case of Smart TV platforms. These cases can lead to additional risk of vulnerabilities due to the need to maintain the security of these multiple TLS deployments and development errors, both at the OS level and across third-party developers.

Assumptions For this study, we assume that when a device uses TLS, the corresponding traffic must be safeguarded to provide authenticity, confidentiality and integrity. Note that we cannot in general know whether the content of any specific TLS connection is sensitive (e.g., contains personal data). When such connections are used to transmit sensitive content such as users' personal data or a manufacturer's confidential machine-learning models, there is a clear need to protect it against attackers. While some TLS endpoints may be public services that exchange non-sensitive data, we cannot *a priori* distinguish such entities, and thus treat all endpoints that use TLS as *potentially sensitive*.

4 METHODOLOGY

This section provides an overview of our methodology, analyses, and how they relate to the research questions.

4.1 Testbed

IoT Devices We study 40 TLS-supporting IoT devices across 6 categories; *Cameras, Smart Hubs, Home Automation, TV, Audio* and *Other Appliances* (Table 1). The testbed is configured to represent a smart home with a wide range of consumer IoT devices connected to the Internet. All devices are located in an isolated space designed to resemble a studio apartment. To interact with devices that support companion apps, we installed and used these apps on smartphones connected to the same network. Network traffic collection is performed at a gateway that provides network access only to our IoT testbed.

We use a software/firmware update discipline that we assume to be typical of an IoT home scenario. Specifically, devices that receive automatic updates are updated at whatever cadence the manufacturer specifies. For devices that require manual intervention for updates, we accepted the updates when explicitly asked by the companion apps of devices. Note that we accept these updated in an ad-hoc manner, and as such, these devices are not regularly updated. We decided to use this approach because (a) we expect many users to also use these devices in a similar way, and (b) getting *all* devices to update at a regular interval could not be automated.

Experimental Setup and Dataset Our study uses a combination of *passive* and *active* experiments. The key difference between the two experiment types is that *active* experiments involve the usage of *mitmproxy* [10] to intercept traffic while *passive* experiments do not. Both experiments need some form of interaction with the IoT devices for generating network activity.

In *passive* experiments we simply record the network traffic generated by devices. This includes data while devices are not in use, and also from interactions with ≈40 consenting study participants enrolled in our IRB-approved study. These participants are members of our academic institution, and are directed simply to use these devices as they please. The passive dataset covers ≈2 years of traffic from January 2018 to March 2020. Among the 40 devices in *passive* experiments, every device generated traffic for at least 6 months, while 32 devices did so for more than 12 months. Passive data allows us to observe the real-world behavior of the devices (a) when they are connected to the network without user interactions, and (b) when users interact with them.

In *active* experiments, we intercept the traffic from our devices by impersonating the server-side of TLS connections. To induce the devices to generate TLS traffic for interception, we leverage the observation that IoT devices generate significant traffic when powered on [52]. Thus we programmatically use TP-Link power plugs to turn devices off and back on again, causing them to boot and potentially establish TLS connections. All 32 devices in *active* experiments generated at least one TLS connection. The bulk of our experiments were performed in March 2021.

Some devices broke, lost manufacturer support or would lose WiFi connectivity until reconfigured again. As a result, such devices did not generate traffic continuously throughout the entirety of our *passive* experimentation period, and were omitted from the *active* experiments (resulting in the discrepancy between number of devices in *active* vs *passive* experiments).

In total, we gathered ≈17M TLS connections (per device average: ≈422K, median: ≈138K connections). Note that our active experiments comprise controlled, repeatable experiments that are conducted without study participants present and represent a snapshot in time (at least 3 minutes after a device reboot). Passive experiments are uncontrolled and they may include participant interactions, thus they enable us to study longitudinal insights across a variety of connections.

4.2 Instrumentation

We use the following instrumentation to gather data for analysis.

TLS handshake analysis (RQ1 and RQ3) To determine whether devices establish secure TLS connections, we extract information about TLS versions and ciphers advertised by clients and selected by servers. We further parse the *ClientHello*s to extract client fingerprints, and use this information to explore the diversity of observed TLS instances.

TLS interception attacks (RQ2) We investigate whether devices are susceptible to several *active* on-path attacks that an adversary can use to compromise TLS connections (Table 2). We picked these attacks because they do not require significant resources (e.g., compromising a root CA, or breaking a cipher using cryptanalysis). They are related to proper certificate chain validation and have previously been found effective against a wide variety of non-browser TLS clients [39], so we extend these to IoT devices. We use *mitmproxy* [10] for performing these attacks.

Note that a potential limitation of our study is that attempts to test vulnerabilities (e.g., using self-signed certificates) will lead to connection errors, and those in turn may cause a device (or some of its functionality) to cease to work, thus suppressing further network connections. To test the potential impact of this issue, we restart devices and repeat all the above attacks with *TrafficPassthrough* where we do not intercept any connections that previously failed when under attack [11]. Encouragingly, we find that *TrafficPassthrough* experiments did not lead to finding *any* new certificate validation failures, even though they produced ≈20.4% more connections (average, in terms of new DNS or TLS hostnames) from these devices. We speculate that these additional connections might be based

Table 2: Overview of the TLS interception attacks.

Attack	Description
NoValidation	Use a self-signed certificate to check whether a device performs any certificate validation.
WrongHostname	Use an unexpired legitimate certificate for a domain under our control to check whether a device performs hostname validation. We send the full chain linking to a trusted root authority during handshake.
InvalidBasicConstraints	Use certificate from the previous attack as a root CA to check whether a device validates *BasicConstraints* extension. We send the full chain linking to a trusted root authority during handshake.

Table 3: Sources for obtaining historical data for CA root certificates trusted by various platforms.

Platform	Total versions	Earliest version year	Comments
Ubuntu	9	2012	We install the *ca-certificates* package and fetch the */etc/ssl/certs/ca-certificates.crt* file from official Docker images.
Android	10	2010	We use version-tagged commits for either */platform /system/ca-certificates* or *luni/src /main/files/cacerts* [15, 16].
Mozilla	47	2013	We extract different file versions from commit history for NSS's *security/nss/lib/ckfw/builtins/certdata.txt* [12].
Microsoft	15	2017	We use the historical information published by Microsoft about its trusted root store certificates [9].

Table 4: Testing our technique for exploring root stores in various TLS libraries. Only two were found to be amenable (shown in italics).

Library	Response for known CA with invalid signature	Response for unknown CA
MbedTLS (v2.21.0)	*Bad Certificate*	*Unknown CA*
OpenSSL (v1.1.1i)	*Decrypt Error*	*Unknown CA*
Oracle Java (v1.8.0)	Certificate Unknown	Certificate Unknown
WolfSSL (v4.1.0)	Bad Certificate	Bad Certificate
GNU TLS (v3.6.15)	No Alert	No Alert
Secure Transport (macOS v11.3)	No Alert	No Alert

Table 5: IoT devices that *downgrade* security upon connection failures (✓ indicates downgrade).

Device	Failed Handshake	Incomplete Handshake	Behavior	Downgraded / Total Destinations
Amazon Echo Dot	✗	✓	Falls back to using SSL 3.0	7 / 9
Amazon Echo Plus	✗	✓	Falls back to using SSL 3.0	6 / 7
Amazon Echo Spot	✗	✓	Falls back to using SSL 3.0	11 / 15
Amazon Fire TV	✗	✓	Falls back to using SSL 3.0	13 / 21
Apple Homepod	✗	✓	Falls back to using TLS 1.0	7 / 9
Google Home Mini	✗	✓	Falls back to supporting a weaker ciphersuite and signature algorithm (TLS_RSA_WITH_3DES_EDE_CBC_SHA and RSA_PKCS1_SHA1)	5 / 5
Roku TV	✓	✓	Falls back from offering 73 ciphersuites to just 1 (TLS_RSA_WITH_RC4_128_SHA)	8 / 15

Table 6: IoT devices that support older TLS versions.

Device	TLS 1.0 Available?	TLS 1.1 Available?
Zmodo Doorbell	✓	✓
Wink Hub 2	✓	✓
Yi Camera	✓	✓
Philips Hub	✓	✓
Smarter Brewer	✓	✓
TP-Link Bulb	✓	✓
Roku TV	✓	✓
Meross Dooropener	✓	✓
LG TV	✓	✓
Google Home Mini	✓	✓
Amazon Fire TV	✓	✓
Amazon Echo Spot	✓	✓
Amazon Echo Plus	✓	✓
Amazon Echo Dot	✓	✓
Amcrest Camera	✓	✓
Samsung Fridge	✗	✓
Samsung Dryer	✗	✓
Wemo Plug	✓	✗

on success responses from some earlier connections (e.g., a login request) and, as such, only appear in *TrafficPassthrough* tests.

Root stores analysis (RQ2) We present a novel technique to detect if a Certificate Authority (CA) root certificate is in the trusted root store of an IoT device. Our key insight is that the TLS protocol specifies different steps for clients when validating a certificate with an *unknown issuer* compared to a certificate with *known issuer but invalid signature*—opening a side channel to infer the presence of trusted root certificates in a client's root store. In this work, we exploit this side channel using TLS *Alert Messages*.

We first use a self-signed root certificate with arbitrary *Subject Name* to intercept a TLS connection originating from the device. The device should fail to establish the connection if it is doing proper certificate validation because our CA certificate is not in its root store. We then intercept the same TLS connection using a *spoofed CA certificate*, i.e., a self-signed root certificate with its *Subject Name*, *Issuer Name* and *Serial Number* matching that of a legitimate root certificate being tested. The client should reject this certificate due to a signature validation error: while the subject name, issuer name, and serial number all match a trusted root certificate, we do not have the root CA's private key to generate a valid signature for the leaf certificate in chain. Thus our interception attempt fails in both cases, but the failure could either be due to the client not recognizing the arbitrary *Subject Name* in its root store, or because it does recognize a *Subject Name* that is in its root store but the leaf certificate has an invalid signature. If we are able to observe this difference in device behavior, we can infer whether a given CA certificate is trusted by the device or not.

We found that the TLS specification provides a mechanism to observe this difference in behavior: per RFC 5246 (TLS 1.2) or RFC 8446 (TLS 1.3), a TLS client may choose to send a TLS *Alert Message* during a connection failure. More specifically, clients can choose to send `unknown_ca` alert to indicate that a trusted CA root certificate could not be found when forming the chain and `decrypt_error` alert to indicate for a signature check failure. For this work, we consider a device amenable to our technique of root store exploration if it sends different alerts based on the type of experiment run.

To realize this experiment, we use the approach from TLS interception attacks to boot devices, intercept their TLS connections, and respond with self-signed certificates as described previously. We then record any TLS *Alert Messages* that appear. It is crucial that a connection from the same TLS instance is triggered from a device every time a root CA is investigated. Otherwise, we cannot know if our exploration is targeted towards one root store or multiple root stores on the same device. For our experiments, our expectation is that devices will follow the same procedure every time they are rebooted.

To obtain a set of CA certificates to spoof, we gathered historical data for CA certificates trusted by various platforms through the sources described in Table 3. We use this data to make two distinct set of certificates:

(1) *Common CA certificates*: we use the latest version of the root store for each platform and extract currently unexpired certificates common to all of them.
(2) *Deprecated CA certificates*: we start with the earliest version of the root store for each platform, and extract all certificates removed from the successor version(s) of the store, but that are currently unexpired. We exclude any certificate if it was once removed but is still present in the latest version of the root store.

Common CA certificates represent the ones trusted by all major (non-IoT) platforms, and thus can be considered likely trustworthy. *Deprecated CA certificates* represent cases where root certificates are retired before expiration, or in some cases explicitly distrusted (e.g., due to noncompliance with CA guidelines), and thus their trustworthiness is (more) questionable. Note that our approach cannot in general reveal all certificates in the root store; rather, it can reveal only those included in our testing set. As such, our analysis may omit non-public root, such as those in private PKIs.

Figure 1: TLS version support for IoT devices. Devices often use multiple versions (rows 2-12), can encounter lack of server support (rows 2-8) and rarely adopt better TLS versions over time (rows 7-10). 28 devices use TLS 1.2 for the vast majority of their advertised and established connections, and are not shown in this figure.

We validated the efficacy of our approach in popular TLS libraries and present results in Table 4. Among the 2/6 libraries that are amenable to this analysis, *MbedTLS* is generally known to be deployed in IoT ecosystems [32]. As we show later in the paper, we empirically found that *OpenSSL* is used by multiple devices that are also amenable to this measurement strategy.

5 RESULTS

We now present the results to answer our research questions.

5.1 TLS Connection Security

In this section, we rely on more than two years of *passively* collected data to study TLS protocol version and ciphersuites, and whether their support improves over time.

Protocol Version As explained earlier, the TLS version used in a connection is determined during the handshake and is based on the highest version supported by both client and server. Since versions prior to 1.2 are deprecated due to security concerns, we focus on the prevalence of such connections in our dataset, and whether their use is due to lack of client and/or server support for newer versions.

Our first observation is good news. A large majority of the devices (28/40) use TLS 1.2 exclusively and are thus not using deprecated versions. However, for other devices, we find a mix of traffic that includes the use of deprecated TLS versions over time.

To visualize this phenomenon and understand how it impacts the security of established connections, in Fig. 1 we visualize a heatmap of the fraction of connections for which each TLS version is *advertised* via *Client Hello*s (left), and *established* via *Server Hello*s (right) over a 2-year period. For each device (y-axis), we use three rows to represent the TLS connections observed over 1.3 (top), 1.2 (middle) or older versions (bottom). Each cell represents the fraction of TLS connections over each TLS version during a particular month of our study (x-axis). Gray cells indicate months where a device did not generate any TLS traffic. Note that Fig. 1 omits the 28 devices that established connections using only TLS 1.2. We make the following observations:

The vast majority of connections happen over TLS 1.2. Only the *Wemo Plug* advertises an insecure TLS version throughout the entire measurement period for all its connections.

Devices tend to support newer protocol versions than the servers they connect to. We find that 32 devices *advertised* support for TLS 1.2 in more than 95% of their connections every month; however, only 24 *established* connections consistently with TLS 1.2. For example, the *LG Dishwasher, Samsung Dryer, Samsung Washer, Samsung Fridge* devices advertise TLS 1.2, and the *Apple Home Pod and Apple TV* devices advertise TLS 1.3, but all of them establish connections using older protocol versions. The finding highlights that the security of TLS connections from IoT devices in many cases is limited by servers rather than the devices themselves.

Devices rarely upgrade to newer protocol versions. The vast majority of devices supported the same TLS versions during the two-year study. The exceptions are *Apple TV* and *Google Home Mini*, which transitioned to using TLS 1.3 (5/2019), and the *Blink Security Hub*, which transitioned to TLS 1.2 (7/2018), for the majority of its advertised connections. (TLS 1.3 was finalized by IETF in 8/2018.)

We do not have ground truth to indicate whether changes in advertised TLS versions are due to TLS software upgrades on the device or due to connections established using a different existing TLS instance on the same device. For the three cases above, we believe they are likely software upgrades because the new protocol versions are used exclusively after the transition. In contrast, the

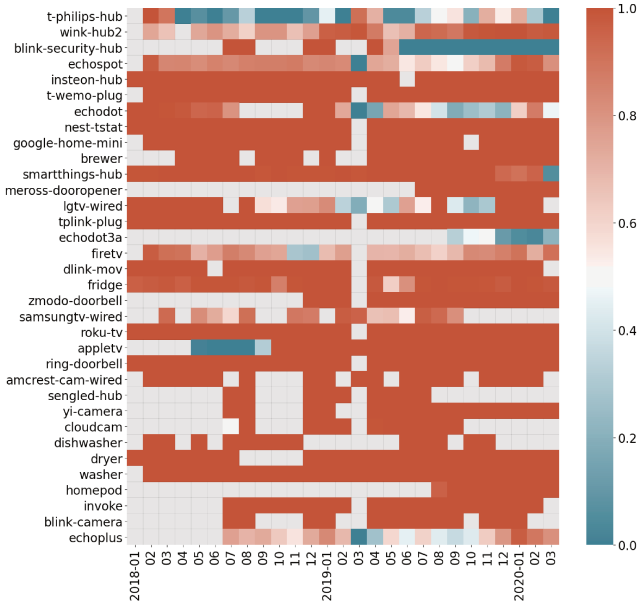

Figure 2: IoT devices that *advertise* handshakes with insecure ciphersuites (lower is better). Most devices do not deprecate these ciphersuites over time. 6 devices rarely advertise such ciphersuites, and are not shown in this figure.

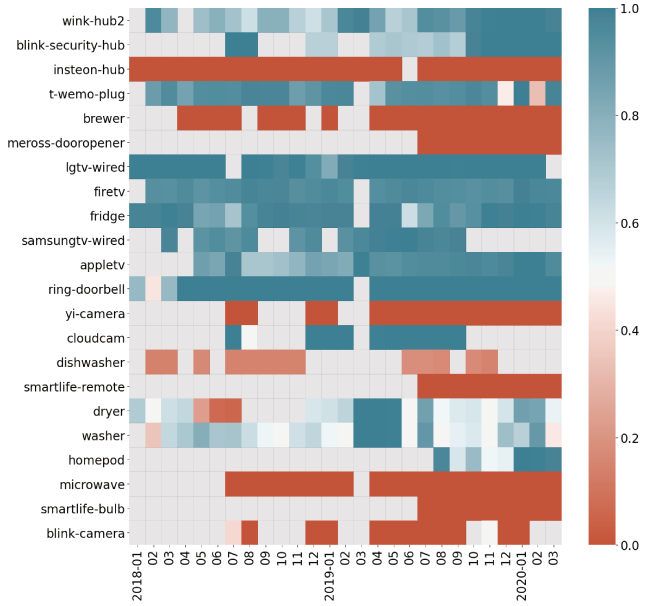

Figure 3: IoT devices that *establish* connections with strong ciphersuites (higher is better). Most devices do not adopt these ciphersuites over time. 18 devices use such ciphersuites for the vast majority of their established connections, and are not shown in this figure.

Insteon Hub appeared to downgrade its advertised and established connections to older TLS versions for a brief period of time (7/2018–8/2019). We manually inspected these cases and found that changes in fractions of connections using older TLS versions were explained by a single set of destinations that were contacted more or less frequently from one month to the next. As such, we do not believe these were due to any TLS software changes. Note, however, that the transition to TLS 1.2 (9/2019) is more likely due to an upgrade in protocol support because older TLS versions are not seen at all after this date.

Devices that advertise multiple maximum TLS versions. We find that 20 devices advertise support for more than one TLS version, with 15 of those advertising multiple maximum versions for the same destinations. This was surprising, since a device with a more secure configuration would advertise only the most recent TLS version as its maximum. There are several potential explanations for this behavior. One explanation could be that different IoT device functionality (e.g., third-party software) uses the same TLS implementation but different configurations. In this case, we hypothesize that connections to different parties would consistently use different TLS configurations. To test this, we labeled each TLS connection as first or third-party using an approach inspired by Ren et al. [52]. We found no patterns that indicate bias toward one TLS version depending on the destination type contacted, and thus we found no evidence to support this hypothesis. Another explanation is that each device contains multiple TLS instances and different software components of a device use them independently. While we do not have any ground truth to confirm it, the observed behavior is consistent with this explanation. We explore this behavior and its implications further in §5.3.

Connection security under attacks The results above focus on connection security observed passively, and only shed light on the maximum advertised protocol version from devices. To better understand the susceptibility of these devices to even weaker security in the face of an active on-path attacker, we conducted active experiments that attempt to force devices to downgrade connection security through connection failures, or negotiate connections with older TLS versions by using them in *ServerHellos*.

We ran experiments using two types of TLS connection failures; *IncompleteHandshake* where we do not reply to a *ClientHello* with *ServerHello*, and *FailedHandshake* where we use a self-signed certificate to cause an unsuccessful handshake. Table 5 lists the 7 devices that downgrade security upon connection failures, the types of handshakes errors that lead to downgrades, how security was downgraded, and how many destinations were susceptible. The most likely reason for such behavior is that clients intentionally want to maximize compatibility with old servers. Interestingly, the majority of—but not all—destinations (i.e., unique domains identified via SNI or DNS) for a device are affected by downgrades. The exception is the *Google Home Mini*, which is susceptible to downgrades on all its connections. The most significant downgrade that we observed was the fallback to SSL 3.0 (which is vulnerable to the POODLE attack) in 4 devices, all from the *Amazon* family.

Next, we investigate which devices support TLS versions older than 1.2 and will establish connections using those older versions, if triggered to do so. Table 6 lists the 19 devices that support TLS

versions older than 1.2. We note that despite the large number of these devices, TLS 1.2 was the most common protocol seen in *established connections* from passive data. As such, the finding highlights that completely protecting against active attackers requires devices to not only advertise TLS 1.2, but also completely disable support for older TLS versions.

Ciphersuites Similar to the protocol version, the selection of a connection's ciphersuite also happens during a connection handshake and depends on client and server compatibility. For a connection to follow best security practices, *strong* ciphersuites that offer forward-secrecy (DHE, ECDHE) should be chosen, while those that are either *insecure* (RC4, DES, 3DES, EXPORT) or do not offer encryption or authentication (ANON, NULL) must be avoided. To study the prevalence and client/server support for these ciphersuites, we plot heatmaps for the advertised and established ciphersuites over time. Each row represents a device, where each cell is the fraction of connections that are insecure (Fig. 2) or strong (Fig. 3) for a given month of the study. As before, gray cells indicate months where there was no TLS traffic from the device. We make the following observations:

Devices never support (ANON, NULL) ciphersuites. We did not observe any TLS connection advertised or established using these.

Devices support weaker ciphersuites than the servers they talk to. 34 devices advertised insecure ciphersuites (Figure 2) but only 2 ever established connections using those (*Wink Hub 2* and *LG TV*). In contrast to support for TLS versions, the devices in our study generally offered to use weaker security than what servers chose to establish.

Devices tend to have better support for perfect forward secrecy than the servers they connect to. 33 devices advertise support for forward secrecy, but a large majority of devices (22) establish most of their connections without it (Fig. 3).

Devices rarely improve usage of ciphersuites over time. Only 2 devices (*Blink Security Hub – 5/2019, SmartThings Hub – 3/2020*) stopped advertising/using weak ciphers during our two-year study (Fig. 2), while 5 (*Apple HomePod – 1/2020, Ring Doorbell – 4/2018, Apple TV – 3/2019, Wink Hub & Blink Security Hub – 10/2019*) adopted perfect forward secrecy (Fig. 3). Surprisingly, Apple TV (10/2018) appeared to increase support for weak ciphers over time.

Devices show varying support for ciphersuites during multiple months. Many devices support insecure ciphersuites in a fraction of their connections as opposed to all or none. Similar to the case with protocol version, the varying support suggests the presence of multiple TLS instances in a device.

Comparison with prior work We now compare TLS versions seen from the IoT devices in our testbeds with those observed in prior work. Note that prior work [41, 43] looked at all traffic from a network provider, not only IoT devices. Specifically, when looking at North American vantage points in November, 2019, a recent study [41] found that ≈60% of client connections support TLS 1.3, while our study found only ≈17% of IoT device connections support TLS 1.3. In April, 2018, Kotzias et al. [43] found that ≈10% connections advertise RC4 ciphersuite support while we find ≈60% of connections do. Relative to other sources of Internet traffic such as browsers, IoT devices and their online infrastructure are slow to adopt modern protocol features and to deprecate insecure ones.

Table 7: IoT devices vulnerable to TLS *interception* attacks. (✓ indicates vulnerability).

Device	No-Validation	InvalidBasic-Constraints	Wrong-Hostname	Vulnerable/Total Destinations
Zmodo Doorbell	✓	✓	✓	6 / 6
Amcrest Camera	✓	✓	✓	2 / 2
Smarter Brewer	✓	✓	✓	1 / 1
Yi Camera	✓	✓	✓	1 / 1
Wink Hub 2	✓	✓	✓	1 / 2
LG TV	✓	✓	✓	1 / 2
Smartthings Hub	✓	✓	✓	1 / 3
Amazon Echo Plus	✗	✗	✓	1 / 8
Amazon Echo Dot	✗	✗	✓	1 / 9
Amazon Echo Spot	✗	✗	✓	1 / 17
Amazon Fire TV	✗	✗	✓	1 / 21

Table 8: Summary of support for different certificate revocation methods among IoT devices.

Method	Devices (Count)
Certificate Revocation Lists (CRLs)	Samsung TV (1)
Online Certificate Status Protocol (OCSP)	Samsung TV, Apple TV, Apple Home Pod (3)
OCSP Stapling	Fire TV, Samsung TV, Echo Spot, Apple Home Pod, Apple TV, Harman Invoke, Echo Dot, Wink Hub 2, Google Home Mini, LG TV, Samsung Fridge, Smartthings Hub (12)

Takeaways Our longitudinal study revealed good and bad news about TLS usage in IoT devices. On the positive side, the IoT devices in our study often rely on TLS1.2 or above, do not support (NULL, ANON) ciphersuites and often support better protocol versions than the servers they connect to. On the negative side, many of the devices in our study do not use the latest protocol version, still support some weak ciphersuites, and tend to not upgrade to modern protocol features over time. Our findings suggest that although most IoT devices establish reasonably secure TLS connections, device manufacturers can improve when it comes to maintaining updated TLS libraries and configurations over time. This will help to reduce their exposure to attacks over time.

5.2 Certificate validation

In this section, we use active experiments to evaluate how well IoT devices validate TLS certificates for the connections they establish. It is important to note that failure to properly validate certificates makes devices susceptible to interception attacks, where the attacker can recover the plaintext content of encrypted connections. To understand the correctness of certificate validation, we test three aspects. First, we identify whether devices are susceptible to interception attacks via the techniques presented in Table 2. Second, we determine whether devices conduct certificate revocation checking. Last, we evaluate our novel probing strategy to reveal the set of trusted root CAs and determine whether devices continue to trust unexpired root certificates that have been deprecated, particularly focusing on *distrusted* certificates.

Table 9: Exploring the root stores of 8 IoT devices. Each cell denotes the number of root certificates present in an IoT device over the number of certificates whose inclusion could be successfully checked.

Device	Common certs (total = 122)	Deprecated certs (total = 87)
Google Home Mini	100% (119/119)	6% (4/71)
Amazon Echo Plus	98% (103/105)	18% (13/72)
Amazon Echo Dot	98% (117/119)	19% (14/72)
Amazon Echo Dot 3	90% (86/96)	27% (17/72)
Wink Hub 2	92% (109/119)	38% (27/72)
Roku TV	91% (96/106)	41% (33/81)
LG TV	93% (96/103)	59% (48/82)
Harman Invoke	82% (67/82)	59% (41/70)

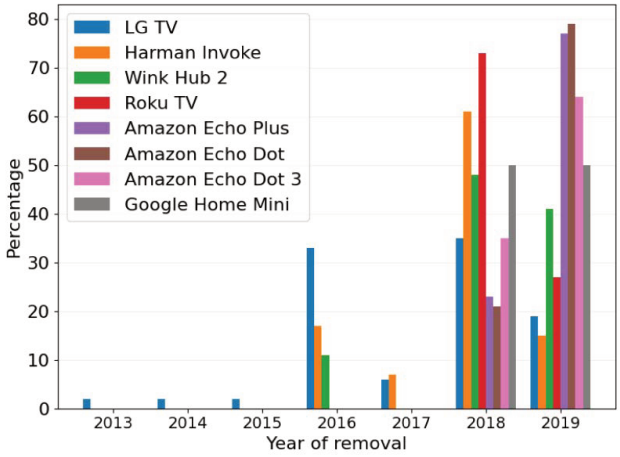

Figure 4: For deprecated CA root certificates still present in IoT devices, we track their year of removal from major platforms.

Invalid certificates We begin by understanding whether devices perform validation correctly when presented with invalid certificates (Table 7). In summary, *seven devices do not perform **any** certificate validation* and are thus vulnerable to traffic interception. Four other devices (all from the *Amazon* family) do not check for correct *Common Name* in certificates and we were thus able to decrypt their TLS traffic using a free certificate obtained from *ZeroSSL* for a domain under our control. Interestingly, the *Yi Camera* disables certification validation completely upon 3 consecutive failed connections.

Through manual inspection of successfully intercepted TLS connections, we found that 7/11 devices transmitted potentially sensitive data to first-party destinations (e.g., "encrypt_key" for *Zmodo Doorbell*, "command server" for *Amcrest Camera*, "deviceSecret" for *LG TV* and "bearer" authentication tokens for *Amazon* devices). This provides strong evidence that lack of certificate validation can have implications for user and device security/privacy.

Interestingly, we also found that 7/11 vulnerable devices (Table 7, column 5) initiated TLS connections to other first or third-party destinations that were *not* vulnerable (likely due to the presence of multiple TLS instances—we explore this behavior and its implications further in §5.3).

Revocation Checking An important aspect of establishing secure connections is for clients to determine whether the server certificate for a connection has been revoked. To test whether devices perform such checks, we use passive data to look for communication with standard revocation endpoints (CRLs, OSCP servers), requests for OCSP staples in *ClientHello*s and presence of *Must Staple* extension in certificates. We find that a large majority of devices (28) do not ever conduct certificate revocation checks, and thus only 12 devices ever attempt to check for revocation for any of the certificates received throughout the measurement period (Table 8). Of those devices, 11 support OCSP Stapling but never encounter a certificate with a *Must Staple* extension. We conclude that the IoT ecosystem provides only limited support for revocation checking, similar to what has been observed by prior work in desktop and mobile browsers [44].

Root Stores When devices continue to trust deprecated or distrusted (and unexpired) CA certificates, they can become susceptible to interception attacks against all destinations if an attacker obtains the corresponding secret key. We now investigate the extent to which IoT devices are vulnerable to this issue.

We use the methodology introduced in §4.1 to detect the inclusion of deprecated-yet-unexpired root store CAs in IoT devices. We excluded appliances not suitable for repeated reboots (i.e., *Washer, Dryer, Thermostat, Fridge*) and the devices that did not validate certificates in any of their TLS connections. For 8/24 remaining devices in the testbed, our methodology successfully triggered different *Alert Messages* to enable root stores exploration.

A summary of the results is provided in Table 9. In some cases, our experiments were inconclusive in determining the inclusion of a particular certificate (e.g., if the device did not generate any traffic). We exclude such cases and present the total number of certificate inclusions divided by the total number of successful experiments for each device in the table. We find the majority of unexpired certificates common to all platforms to be present in all devices probed (second column in the table). This is good news, as it suggests that IoT devices, web browsers, and OSes trust a similar set of (presumably trustworthy) CA certificates.

Interestingly, however, all devices also contain at least one deprecated-yet-unexpired root certificate, i.e., that has already been removed from one or more major platforms. With the exception of the Google Home Mini, the IoT devices we tested contain significant fractions (if not a majority) of root certificates that were deprecated from other platforms.

To understand how long such deprecated-yet-unexpired root certificates remain in device root stores, we plot the *staleness* of each root certificate in terms of the year it was removed from one of the four reference platforms in Figure 4. (If a certificate was removed from multiple stores, we use the latest year of removal.) Devices with large numbers of certificates that were removed years ago are either not updating their root stores or not interested in deprecating certificates.

We find that the majority of observed stale root certificates were deprecated in the years 2018 and 2019, likely biased by the fact that

the devices were manufactured at or shortly before those years. Surprisingly, we find that one device (*LG TV*) contains unexpired root CAs that were deprecated as early as 2013. We note that the devices in our testbed were able to receive regular updates during our study. More specifically, *LG TV* was last updated in July 2019 and *Roku TV* in September 2020, while the bulk of our experiments were performed in 2021. Other devices such as voice assistants from Google and Amazon receive updates automatically as long as they are connected to the Internet. This suggests that some manufacturers are not updating root stores at the same cadence (if at all) as other software updates.

A root certificate that is deprecated is not necessarily untrusted, as some may be removed for "administrative" reasons such as regular key rotations (e.g., [1]). However, the *TurkTrust* (2013) and *Certinomis* (2019) CAs were explicitly distrusted by Mozilla while *CNNIC* (2015) and *WoSign* (2016) are in the Google blocklist due to a failure to comply with CA guidelines (e.g., *TurkTrust* was responsible for an unauthorized certificate for `google.com`) [2, 13, 17]. Arguably these root certificates should not be trusted by any devices. Surprisingly, we found that one or more of these CAs explicitly distrusted by various platforms were *still trusted by all devices*. The fact that these root certificates remain trusted by devices can open them to arbitrary interception attacks if the private key for those certificates were shared with adversaries (*WoSign* incident [7]).

We note that these IoT devices tend to contact a small set of destinations, but nonetheless contain root stores used by web browsers/OSes that are expected to contact arbitrary destinations. An important question is whether these devices all need to use such large root stores, or instead some of the devices can reduce their trusted set of certificates to cover only the destinations that are required for the device.

Takeaways 28 IoT devices show some form of certificate validation limitations. Some devices skip certificate validation altogether and most do not bother to check for revoked certificates. All of the affected TLS connections were contacting first-party destinations. We conclude that even popular IoT devices from major manufacturers exhibit poor TLS validation for at least some of their connections. Further, all of the devices that we could successfully probe for root certificates contained at least one that was deprecated and distrusted, despite the fact that the devices themselves install regular updates. These deprecated CAs root certificates—particularly ones that are distrusted—can be perceived as the weakest link in TLS security for IoT devices.

5.3 Diversity of TLS Behavior

In this section, we explore the diversity of TLS behaviors observed for individual devices and across devices. The goal of this analysis is to shed light on how IoT devices use shared or different TLS implementations and configurations, and the potential ramifications on security.

Our primary investigative tool is *TLS fingerprinting*; namely, we generate TLS fingerprints for 32 devices and compare them to a publicly available database of 1,684 fingerprints that covers a wide variety of sources such as different browsers, multiple versions of TLS libraries, and malware samples [43]. Each fingerprint is labeled

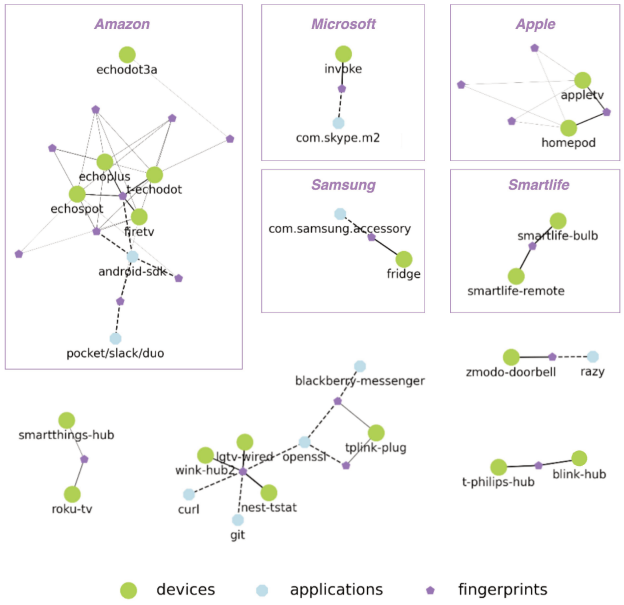

Figure 5: IoT devices that likely share TLS libraries with other devices and applications.

with the *application* that generated it (e.g., *OpenSSL, curl, android-sdk*). We generate fingerprints for the TLS connections from our devices in the same way as done during the database compilation. Since devices can update their libraries and that may affect the corresponding fingerprints, here we only study TLS traffic from active experiments that represent a snapshot in time.

Devices with more than one TLS fingerprint. For 18/32 IoT devices in our experiments, we found only a single TLS fingerprint per device, likely due to the use of a single TLS instance. This can simplify TLS security management by having only one instance to maintain. However, we found that 14/32 devices had connections with more than one fingerprint, indicating the presence of multiple instances. This can help explain the mixed support for TLS connection security and the presence of TLS interception vulnerabilities in some (but not all) connections from a device.

While we do not know why there are multiple TLS instances on a single device (because we lack access to firmware for these devices), one conjecture is that these devices may contain different first- and third-party components, each using different TLS instances. These components can come from a variety of sources such as user-installed software (e.g., app stores) or the usage of multiple frameworks during software development (e.g., Golang, Java, and Python come pre-bundled with different TLS instances). If true, such behavior can make it harder to maintain TLS security over time, as both device manufacturers and other developers need to secure and maintain all of these instances.

TLS fingerprints shared across devices. We find that 19 devices share at least one TLS fingerprint with *other* devices and/or applications (e.g., *OpenSSL*). This is likely because multiple devices share the same (and in many cases, open source) TLS library.

To better understand the nature of shared TLS instances, we produced a graph of devices and applications with the same fingerprints. There are three types of nodes in the graph: devices (from our study) and applications (from Kotzias et al.[43]) that generate TLS fingerprints, and the set of unique fingerprints that are shared among them. Edges between a device/application and fingerprint indicate that we observed a device or application using that fingerprint. Figure 5 visualizes this graph. In the figure, the thicker edges correspond to the most-used fingerprint (and likely, the most-used TLS instance) for each device. Note that the graph includes an edge only if the TLS fingerprint it connects to is shared with at least one other node, i.e., all non-shared fingerprints and edges are removed from the figure to improve readability. Dashed edges represent a fingerprint shared with a labeled application from Kotzias et al.[43], and thus they do not represent observed traffic in our study.

Our first observation is that devices and applications from the same manufacturer share fingerprints—this can be observed with labeled clusters (e.g., *Amazon, Microsoft*, and *Apple*). It is not surprising that these devices are likely using the same TLS instances, but it nonetheless could be good news for maintaining security because it indicates that the manufacturer likely needs to maintain one set of TLS instances across devices. These shared instances also suggest that many of our findings apply to other devices belonging to the same manufacturers that are not in our testbed.

Our next observation is about devices that share fingerprints with applications in the fingerprint database. For example, the dominant fingerprint from Amazon Fire TV is the same as one from *android-sdk*, and we verified that the device runs a fork of Android OS [6]. Similarly, six devices exhibit the same TLS fingerprints as the *OpenSSL* library, likely indicating that *OpenSSL* is used on those devices. This helps to explain why our technique for root stores exploration worked for *Invoke, LG TV, and Wink Hub 2*: despite being produced by different manufacturers, they all share fingerprints with *OpenSSL*—one of the two libraries we found amenable to the root stores exploration technique.

While we pointed out above that shared TLS instances can be good in the sense that they are easier to maintain, sharing can also be a double-edged sword. Specifically, a security vulnerability in one TLS instances can immediately impact large numbers of devices. For example, in the TLS certificate validation analysis, we found that *Amazon* devices fall back to TLS 1.0 during a downgrade attack. The TLS fingerprinting analysis shows that this is likely because they share the same vulnerable implementation. (Interestingly, the *Echo Dot 3* is the only *Amazon* device in our testbed not susceptible to the downgrade attack, and its fingerprints have smaller overlap with those from other *Amazon* devices.) Importantly, our observations hint at a way for an attacker to scale attacks by identifying and exploiting vulnerable TLS implementations that are shared among multiple devices.

Takeaways IoT devices show similarity of TLS fingerprints with (*i*) other devices from the same manufacturer (e.g., all *Amazon* devices), and (*ii*) various TLS clients (e.g., *LG TV* and *Wink Hub 2* with *OpenSSL*)—suggesting that our findings apply to many more devices not tested in our experiments, and that security vulnerabilities found in one instance can affect large numbers of devices. We also found that multiple TLS instances are deployed in the same device in many cases, potentially making it difficult to maintain TLS security over time.

6 DISCUSSION

Recommendations Client support for TLS security has been an underexplored area in recent research. Our findings, however, paint a complex picture of connection security and certificate validation in connections from IoT devices. For instance, some devices support the latest secure TLS features but still negotiate weak connections due to lack of server support. Similarly, some devices fail to validate certificates, but only for some connections. Device root stores are infrequently updated (if at all), and several devices likely include multiple TLS instances.

The user risks due to insecure/incorrect TLS implementations in their IoT devices are similar to the risks for any other systems using TLS, such as web browsers and other apps. For example, MITM attacks may be carried out *not only* by any on-path attackers (e.g., a malicious router), but by other devices on the same user network as well, such as a malicious IoT device using ARP spoofing. If the attack is successful, it can expose potentially sensitive user data, such as microphone data from a smart speaker or login credentials.

To mitigate this, our key recommendation to consumer IoT device manufacturers is to audit, upgrade and maintain their devices' TLS instances in a consistent and uniform way that safeguards all of their network traffic. One way to do this is to provide TLS as an operating system service (i.e., POSIX socket call) as proposed by O'Neill et al. [48]. Multiple components within a device, and multiple devices in the IoT ecosystem can then use the service to enable TLS in a consistent way. In a similar vein, we encourage industry groups like the IoxT alliance [8] to incorporate TLS security standards into their guidelines for manufacturers to follow, as well as verification tests. In fact, the IoxT alliance can also join the CA/Browser Forum consortium [4] to adopt the same standards as web browsers when it comes to trust in root certificates.

IoT devices can also rely on certificate pinning, a technique to mandate the use of particular certificates in the chain sent by a server, to mitigate some of the vulnerabilities found in our study. More specifically, the interception attacks we presented (Table 7) could have been prevented with the proper use of certificate pinning. But it is important to highlight that certificate pinning is not a panacea—pinning can help only in cases of compromised root stores if the leaf certificate is pinned (rather than the root). Further, certificate validation checks are necessary even if pinning is implemented. Otherwise, devices might appear secure but will remain susceptible to sophisticated MITM attacks (e.g., [40]).

An internal or third-party auditing service can also help IoT vendors keep their TLS instances up-to-date with the evolving security recommendations. IoT devices can be configured to create TLS connections to the auditing service at regular intervals (e.g., once every reboot). The service can then audit the security of the connections (e.g., ciphersuites offered by the device during handshake). As new attacks are discovered, the service can contact manufacturers to alert them about new vulnerabilities and mitigations.

Another possible mitigation strategy that IoT users can use is to interpose a trusted network component between their IoT devices and the Internet, similar to the one proposed by Hesselman et al.

[40], to verify that TLS connections are being securely established. If such verification fails, the component pauses the connection and reports the issue to the user, which is left with the choice whether to allow the insecure TLS connection or not, as it happens for web browsers.

Limitations Our study had several limitations. First, we chose a limited number of devices to make the scope of our experiments practical. As such, our results are biased by the selection of (a) popular consumer devices, and (b) multiple devices from the same manufacturer. Second, our choice of TLS interception attacks reflected the ones that are easily exploitable by an in-network adversary. Other sophisticated attacks that use cryptanalysis on a sufficiently large amount of network traffic (e.g., POODLE, SWEET32) are difficult to mount (e.g., need JavaScript injection to repeatedly trigger requests) but could nonetheless compromise TLS security in some IoT devices. Third, the coverage of our analyses could be improved by (*i*) relying on techniques from other works to automate device interactions (e.g., using smartphones [45], reverse-engineering exposed APIs [55]), and (*ii*) inspecting source-code when possible (e.g., firmware extraction from memory, rooting Android-based devices, crawling third-party marketplaces).

Unfortunately, all these techniques require device-specific efforts and do not generally scale well to other devices. Finally, our technique to explore root stores does not generalize for all devices. One reason is that some implementations choose to not send any TLS alerts over connection failures. Moreover, unlike TLS 1.2, which mandated the usage of "appropriate" alerts on encountering fatal errors, TLS 1.3 made it optional. This motivates the need to search for better techniques to exploit the side-channel and explore root stores in more IoT devices.

Responsible disclosure We contacted manufacturers of the 11 IoT devices to responsibly disclose our successful *interception* attacks (Table 7). Unlike other devices that showed weaknesses due to stale root stores or compatibility with older protocol versions and weaker ciphersuites, these devices had vulnerabilities severe enough that we were able to actively exploit them and extract decrypted TLS communications from their first-party connections.

Unfortunately, one vendor categorized the issue as "SSL/TLS best practices" and as such out-of-scope for their vulnerability disclosure program; two other vendors believed the issue to not be serious as the information disclosed in compromised TLS connections was not sensitive; one vendor mistakenly believed that the issue was due to their choice of using a custom root certificate; and only one vendor confirmed that the issues were fixed by releasing a firmware update. We believe these manufacturer responses reveal a wide range of beliefs about TLS security and how they should be improved. We conclude that despite some initiative by manufacturers to secure the devices such as the aforementioned IoxT alliance, there is still plenty of room for improvement of TLS usage in the IoT ecosystem.

Ethical considerations This study involved human subjects that participated after completing informed consent materials that are part of our IRB-approved study. No personal or sensitive data about individuals is collected as part of this study. The active experiments exploited vulnerabilities only for the devices in our lab, and we did not use any information gleaned from these experiments to attack other devices or cloud services.

We anonymized only the manufacturer responses to our disclosures. When making this decision, we balanced risks and benefits to relevant parties. Namely, we saw no additional risks to consumers with these devices, as our measurement strategy is now public and anyone can reproduce it for any device. However, we see potential benefits to naming vendors. First, any vendors that have not updated devices after the responsible disclosure period might find new incentives to do so given public knowledge of the flaws. Second, consumers with devices that are not updated can use this information to discontinue their use. Further, we believe the research community can benefit from this information to reproduce and extend our work, potentially finding other opportunities to improve IoT security. In this vein, we follow the precedent set by prior work on IoT device security [26] that also revealed names in a similar way, and helped us with reproducibility during our experiments. To summarize, the vulnerabilities have been responsibly disclosed, following community norms, and we believe the benefits of transparency outweigh any additional risks from publicly naming manufacturers after the responsible disclosure period.

7 RELATED WORK

IoT/TLS vulnerability detection Alrawi et al.'s SoK [26] is the closest work to ours regarding the security evaluation of IoT devices. Their work covered 45 devices from four different dimensions; devices themselves along with their cloud endpoints, communication channels and mobile apps. Their analysis was not focused on TLS usage, and despite some overlap, it is different from our work in the following key ways. First, they explored the server-side security of TLS connections by establishing connections to devices or their cloud endpoints; in contrast, we explore the client-side security by analyzing the TLS connections *initiated by devices*. Second, they used self-signed certificates to assess device validation of certificates, while our analyses rely on more techniques with invalid certificates and also explore the CA root certificates trusted by devices. And third, their analysis represented a snapshot in time while we use passive data to explore 2-year longitudinal trends of TLS usage.

Longitudinal TLS measurements There is a significant body of research on analyzing TLS usage from different vantage points i.e., passive monitoring of university networks [41–43], server-side connection logs [35], active Internet scans [27, 33, 34], browser telemetry data [37], and Android usage statistics [51]. In this work, we study longitudinal TLS usage from a vantage point missing in prior work: traffic from IoT devices in a simulated smart home.

Root store analysis To the best of our knowledge, Fadai et al. [36] is the only work to have investigated the historical data for Mozilla's trusted certificates. They evaluated the trust implications of root certificates from several platforms in terms of the owner status (i.e., private entity or governmental organization) and country of origin. Other works proposed techniques to restrict the set of root CAs trusted by users based on the insights that (*i*) CAs commonly sign a handful of top-level domains [42], (*ii*) some CAs have not signed any certificates used by the HTTPS servers [50], and (*iii*) unique browsing history enables individualization of the

trusted CAs set [30]. Some works have also focused on the user-trusted certificates present in the wild and that do not belong to audited root stores—Vallina-Rodriguez et al. [54] explored vendor and app-specific additions to the official Android root store, and Durumeric et al. [35] explored the additions due to middleboxes such as an antivirus software or a corporate proxy. In this work, we explore the root stores of IoT devices where inspection is difficult due to their blackbox nature.

TLS Fingerprinting TLS fingerprinting has been used frequently in the past to infer client behaviors – from detecting malware [22] to the usage of censorship circumvention tools [38] and client identification [35, 41, 43, 51]. In this work, we explore how TLS fingerprinting sheds light on some of our findings in a setting where a wide variety of IoT devices are available, and network traffic may originate from multiple clients and networking libraries within the same device.

8 CONCLUSION

This paper filled an important knowledge gap in our understanding of TLS behavior from consumer IoT devices using more than two years passive measurements along with active experiments to reveal TLS vulnerabilities. We find a wide range of security-related TLS behaviors ranging from good (a large majority of tested devices use TLS 1.2 or higher), to bad (more than half of the devices advertise deprecated TLS versions or insecure ciphersuites in a significant fraction of their connections), and critically flawed (11 devices are vulnerable to TLS interception attacks because they do not properly validate server certificates). Further, we find that devices are slow to adopt new TLS versions and to secure the set of supported ciphersuites, and they also rarely remove deprecated and distrusted CA certificates from their root stores. Finally, we used TLS fingerprinting to identify cases where individual devices use multiple distinct TLS instances, and those where different devices use the same TLS instances—each with implications for security, e.g., shared vulnerabilities that can facilitate attack scaling. We conclude that TLS clients in IoT devices have much room for improvement, and we recommend that manufacturers adopt uniformly secure TLS instances and industry standards [8], and conduct regular auditing and updating to ensure their devices' connections remain secure.

To ensure reproducibility and enable new research, we have made all of our longitudinal TLS handshake data, controlled experimentation data and analysis software publicly available at: https://github.com/NEU-SNS/IoTLS.

9 ACKNOWLEDGMENTS

We thank the anonymous reviewers and our shepherd, Karyn Benson, for their helpful feedback. This research was supported by the following grants: NSF (BehavIoT CNS-1909020, ProperData SaTC-1955227), EU's H2020 Program (TRUST aWARE, Grant Agreement No. 101021377), Spanish National Grant ODIO (PID2019-111429RB-C22) and Consumer Reports (Digital Lab Fellowship for Daniel J. Dubois).

REFERENCES

[1] [n. d.]. 1493822 - Removal of "Visa eCommerce Root" CA from Mozilla Root Program. https://bugzilla.mozilla.org/show_bug.cgi?id=1493822. ([n. d.]). (Accessed on 05/16/2021).
[2] [n. d.]. 1552374 - Remove Certinomis - Root CA. https://bugzilla.mozilla.org/show_bug.cgi?id=1552374. ([n. d.]). (Accessed on 05/16/2021).
[3] [n. d.]. Apple, Google, Microsoft, and Mozilla come together to end TLS 1.0 | Ars Technica. https://arstechnica.com/gadgets/2018/10/browser-vendors-unite-to-end-support-for-20-year-old-tls-1-0/. ([n. d.]). (Accessed on 05/14/2021).
[4] [n. d.]. CAB Forum | Certification Authorities, Web Browsers, and Interested Parties Working to Secure the Web. https://cabforum.org/. ([n. d.]). (Accessed on 09/27/2021).
[5] [n. d.]. ELIMINATING_OBSOLETE_TLS_UOO197443-20.PDF. https://media.defense.gov/2021/Jan/05/2002560140/-1/-1/0/ELIMINATING_OBSOLETE_TLS_UOO197443-20.PDF. ([n. d.]). (Accessed on 05/25/2021).
[6] [n. d.]. Fire OS Overview | Amazon Fire TV. https://developer.amazon.com/docs/fire-tv/fire-os-overview.html. ([n. d.]). (Accessed on 11/21/2020).
[7] [n. d.]. Google Online Security Blog: Distrusting WoSign and StartCom Certificates. https://security.googleblog.com/2016/10/distrusting-wosign-and-startcom.html. ([n. d.]). (Accessed on 05/26/2021).
[8] [n. d.]. ioXt - The Global Standard for IoT Security. https://www.ioxtalliance.org/. ([n. d.]). (Accessed on 05/26/2021).
[9] [n. d.]. Microsoft Trusted Root Certificate Program: Participants - TechNet Articles - United States (English) - TechNet Wiki. https://social.technet.microsoft.com/wiki/contents/articles/31634.microsoft-trusted-root-certificate-program-participants.aspx. ([n. d.]). (Accessed on 05/19/2021).
[10] [n. d.]. mitmproxy - an interactive HTTPS proxy. https://mitmproxy.org/. ([n. d.]). (Accessed on 05/26/2021).
[11] [n. d.]. mitmproxy/tls_passthrough.py at main · mitmproxy/mitmproxy. https://github.com/mitmproxy/mitmproxy/blob/main/examples/contrib/tls_passthrough.py. ([n. d.]). (Accessed on 05/26/2021).
[12] [n. d.]. mozilla-central: certdata.txt. https://hg.mozilla.org/mozilla-central/file/tip/security/nss/lib/ckfw/builtins/certdata.txt. ([n. d.]). (Accessed on 05/19/2021).
[13] [n. d.]. net/data/ssl/blocklist - chromium/src - Git at Google. https://chromium.googlesource.com/chromium/src/+/refs/heads/main/net/data/ssl/blocklist/. ([n. d.]). (Accessed on 05/26/2021).
[14] [n. d.]. Number of IoT devices 2015-2025 | Statista. https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/. ([n. d.]). (Accessed on 12/02/2020).
[15] [n. d.]. platform/libcore - Git at Google. https://android.googlesource.com/platform/libcore/. ([n. d.]). (Accessed on 05/19/2021).
[16] [n. d.]. Refs - platform/system/ca-certificates - Git at Google. https://android.googlesource.com/platform/system/ca-certificates/+refs. ([n. d.]). (Accessed on 05/19/2021).
[17] [n. d.]. Revoking Trust in Two TurkTrust Certificates - Mozilla Security Blog. https://blog.mozilla.org/security/2013/01/03/revoking-trust-in-two-turktrust-certficates/. ([n. d.]). (Accessed on 05/16/2021).
[18] [n. d.]. rfc2818. https://datatracker.ietf.org/doc/html/rfc2818. ([n. d.]). (Accessed on 05/22/2021).
[19] [n. d.]. rfc5280. https://datatracker.ietf.org/doc/html/rfc5280. ([n. d.]). (Accessed on 05/22/2021).
[20] [n. d.]. This POODLE Bites: Exploiting The SSL 3.0 Fallback. https://www.openssl.org/~bodo/ssl-poodle.pdf. ([n. d.]). (Accessed on 05/03/2021).
[21] [n. d.]. TLS Cipher String · OWASP Cheat Sheet Series. https://web.archive.org/web/20190716105553/https://cheatsheetseries.owasp.org/cheatsheets/TLS_Cipher_String_Cheat_Sheet.html. ([n. d.]). (Accessed on 05/16/2021).
[22] [n. d.]. TLS Fingerprinting in the Real World - Cisco Blogs. https://blogs.cisco.com/security/tls-fingerprinting-in-the-real-world. ([n. d.]). (Accessed on 11/25/2020).
[23] [n. d.]. What You Need To Know About the SolarWinds Supply-Chain Attack | SANS Institute. https://www.sans.org/blog/what-you-need-to-know-about-the-solarwinds-supply-chain-attack/. ([n. d.]). (Accessed on 04/04/2021).
[24] David Adrian, Karthikeyan Bhargavan, Zakir Durumeric, Pierrick Gaudry, Matthew Green, J Alex Halderman, Nadia Heninger, Drew Springall, Emmanuel Thomé, Luke Valenta, et al. 2015. Imperfect forward secrecy: How Diffie-Hellman fails in practice. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. 5–17.
[25] Nadhem J AlFardan, Daniel J Bernstein, Kenneth G Paterson, Bertram Poettering, and Jacob CN Schuldt. 2013. On the Security of RC4 in TLS and WPA. In *USENIX Security Symposium*. 173.
[26] Omar Alrawi, Chaz Lever, Manos Antonakakis, and Fabian Monrose. 2019. SoK: Security Evaluation of Home-Based IoT Deployments. In *2019 IEEE Symposium on Security and Privacy (SP)*. 1362–1380. https://doi.org/10.1109/SP.2019.00013

[27] Johanna Amann, Oliver Gasser, Quirin Scheitle, Lexi Brent, Georg Carle, and Ralph Holz. 2017. Mission accomplished? HTTPS security after DigiNotar. In *Proceedings of the 2017 Internet Measurement Conference*. 325–340.

[28] Benjamin Beurdouche, Karthikeyan Bhargavan, Antoine Delignat-Lavaud, Cédric Fournet, Markulf Kohlweiss, Alfredo Pironti, Pierre-Yves Strub, and Jean Karim Zinzindohoue. 2015. A messy state of the union: Taming the composite state machines of TLS. In *2015 IEEE Symposium on Security and Privacy*. IEEE, 535–552.

[29] Karthikeyan Bhargavan and Gaëtan Leurent. 2016. On the practical (in-) security of 64-bit block ciphers: Collision attacks on HTTP over TLS and OpenVPN. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. 456–467.

[30] Johannes Braun and Gregor Rynkowski. 2013. The Potential of an Individualized Set of Trusted CAs: Defending against CA Failures in the Web PKI. In *2013 International Conference on Social Computing*. 600–605. https://doi.org/10.1109/SocialCom.2013.90

[31] Chad Brubaker, Suman Jana, Baishakhi Ray, Sarfraz Khurshid, and Vitaly Shmatikov. 2014. Using frankencerts for automated adversarial testing of certificate validation in SSL/TLS implementations. In *2014 IEEE Symposium on Security and Privacy*. IEEE, 114–129.

[32] Sze Yiu Chau, Omar Chowdhury, Endadul Hoque, Huangyi Ge, Aniket Kate, Cristina Nita-Rotaru, and Ninghui Li. 2017. Symcerts: Practical symbolic execution for exposing noncompliance in X. 509 certificate validation implementations. In *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 503–520.

[33] Zakir Durumeric, James Kasten, Michael Bailey, and J Alex Halderman. 2013. Analysis of the HTTPS certificate ecosystem. In *Proceedings of the 2013 conference on Internet measurement conference*. 291–304.

[34] Zakir Durumeric, Frank Li, James Kasten, Johanna Amann, Jethro Beekman, Mathias Payer, Nicolas Weaver, David Adrian, Vern Paxson, Michael Bailey, and J. Alex Halderman. 2014. The Matter of Heartbleed. In *Proceedings of the 2014 Conference on Internet Measurement Conference (IMC '14)*. Association for Computing Machinery, New York, NY, USA, 475–488. https://doi.org/10.1145/2663716.2663755

[35] Zakir Durumeric, Zane Ma, Drew Springall, Richard Barnes, Nick Sullivan, Elie Bursztein, Michael Bailey, J Alex Halderman, and Vern Paxson. 2017. The Security Impact of HTTPS Interception.. In *NDSS*.

[36] Tariq Fadai, Sebastian Schrittwieser, Peter Kieseberg, and Martin Mulazzani. 2015. Trust me, I'm a Root CA! Analyzing SSL Root CAs in Modern Browsers and Operating Systems. In *2015 10th International Conference on Availability, Reliability and Security*. IEEE, 174–179.

[37] Adrienne Porter Felt, Richard Barnes, April King, Chris Palmer, Chris Bentzel, and Parisa Tabriz. 2017. Measuring HTTPS adoption on the web. In *26th USENIX Security Symposium (USENIX Security 17)*. 1323–1338.

[38] Sergey Frolov and Eric Wustrow. 2019. The use of TLS in Censorship Circumvention.. In *NDSS*.

[39] Martin Georgiev, Subodh Iyengar, Suman Jana, Rishita Anubhai, Dan Boneh, and Vitaly Shmatikov. 2012. The most dangerous code in the world: validating SSL certificates in non-browser software. In *Proceedings of the 2012 ACM conference on Computer and communications security*. 38–49.

[40] Cristian Hesselman, Jelte Jansen, Marco Davids, and Ricardo de O Schmidt. 2017. *SPIN: a user-centric security extension for in-home networks*. Technical Report. SIDN Labs Technical report SIDN-TR-2017-002.

[41] Ralph Holz, Jens Hiller, Johanna Amann, Abbas Razaghpanah, Thomas Jost, Narseo Vallina-Rodriguez, and Oliver Hohlfeld. 2020. Tracking the deployment of TLS 1.3 on the Web: A story of experimentation and centralization. *ACM SIGCOMM Computer Communication Review* 50, 3 (2020), 3–15.

[42] James Kasten, Eric Wustrow, and J Alex Halderman. 2013. CAge: Taming certificate authorities by inferring restricted scopes. In *International Conference on Financial Cryptography and Data Security*. Springer, 329–337.

[43] Platon Kotzias, Abbas Razaghpanah, Johanna Amann, Kenneth G. Paterson, Narseo Vallina-Rodriguez, and Juan Caballero. 2018. Coming of Age: A Longitudinal Study of TLS Deployment. In *Proceedings of the Internet Measurement Conference 2018 (IMC '18)*. Association for Computing Machinery, New York, NY, USA, 415–428. https://doi.org/10.1145/3278532.3278568

[44] Yabing Liu, Will Tome, Liang Zhang, David Choffnes, Dave Levin, Bruce Maggs, Alan Mislove, Aaron Schulman, and Christo Wilson. 2015. An End-to-End Measurement of Certificate Revocation in the Web's PKI. In *Proceedings of the 2015 Internet Measurement Conference (IMC '15)*. Association for Computing Machinery, New York, NY, USA, 183–196. https://doi.org/10.1145/2815675.2815685

[45] Anna Maria Mandalari, Daniel J. Dubois, Roman Kolcun, Muhammad Talha Paracha, Hamed Haddadi, and David Choffnes. 2021. Blocking without Breaking: Identification and Mitigation of Non-Essential IoT Traffic. In *Proc. of the Privacy Enhancing Technologies Symposium (PETS)*.

[46] Bodo Möller, Thai Duong, and Krzysztof Kotowicz. 2014. This POODLE bites: exploiting the SSL 3.0 fallback. *Security Advisory* (2014).

[47] Marten Oltrogge, Nicolas Huaman, Sabrina Amft, Yasemin Acar, Michael Backes, and Sascha Fahl. 2021. Why Eve and Mallory Still Love Android: Revisiting TLS (In) Security in Android Applications. In *30th USENIX Security Symposium (USENIX Security 21)*.

[48] Mark O'Neill, Scott Heidbrink, Jordan Whitehead, Tanner Perdue, Luke Dickinson, Torstein Collett, Nick Bonner, Kent Seamons, and Daniel Zappala. 2018. The Secure Socket API: TLS as an Operating System Service. In *27th USENIX Security Symposium (USENIX Security 18)*. 799–816.

[49] Damilola Orikogbo, Matthias Büchler, and Manuel Egele. 2016. CRiOS: Toward large-scale iOS application analysis. In *Proceedings of the 6th Workshop on Security and Privacy in Smartphones and Mobile Devices*. 33–42.

[50] Henning Perl, Sascha Fahl, and Matthew Smith. 2014. You Won't Be Needing These Any More: On Removing Unused Certificates from Trust Stores. In *International Conference on Financial Cryptography and Data Security*. Springer, 307–315.

[51] Abbas Razaghpanah, Arian Akhavan Niaki, Narseo Vallina-Rodriguez, Srikanth Sundaresan, Johanna Amann, and Phillipa Gill. 2017. Studying TLS usage in Android apps. In *Proceedings of the 13th International Conference on emerging Networking EXperiments and Technologies*. 350–362.

[52] Jingjing Ren, Daniel J. Dubois, David Choffnes, Anna Maria Mandalari, Roman Kolcun, and Hamed Haddadi. 2019. Information Exposure for Consumer IoT Devices: A Multidimensional, Network-Informed Measurement Approach. In *Proc. of the Internet Measurement Conference (IMC)*.

[53] Said Jawad Saidi, Anna Maria Mandalari, Roman Kolcun, Hamed Haddadi, Daniel J Dubois, David Choffnes, Georgios Smaragdakis, and Anja Feldmann. 2020. A Haystack Full of Needles: Scalable Detection of IoT Devices in the Wild. In *Proceedings of the ACM Internet Measurement Conference*. 87–100.

[54] Narseo Vallina-Rodriguez, Johanna Amann, Christian Kreibich, Nicholas Weaver, and Vern Paxson. 2014. A Tangled Mass: The Android Root Certificate Stores. In *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*. 141–148.

[55] Janus Varmarken, Hieu Le, Anastasia Shuba, Athina Markopoulou, and Zubair Shafiq. 2020. The TV is Smart and Full of Trackers: Measuring Smart TV Advertising and Tracking. *Proceedings on Privacy Enhancing Technologies* 2020, 2 (2020).