Validating Security Protocols with Cloud-Based Middleboxes

Curtis R. Taylor and Craig A. Shue
Worcester Polytechnic Institute
{crtaylor, cshue}@cs.wpi.edu

***Abstract*—Residential networks pose a unique challenge for security since they are operated by end-users that may not have security expertise. Residential networks are also home to devices that may have lackluster security protections, such as Internet of Things (IoT) devices, which may introduce vulnerabilities. In this work, we introduce TLSDeputy, a middlebox-based system to protect residential networks from connections to inauthentic TLS servers. By combining the approach with OpenFlow, a popular software-defined networking protocol, we show that we can effectively provide residential network-wide protections across diverse devices with minimal performance overheads.**

I. Introduction

Residential networks pose unique challenges for the security community. While these networks are ubiquitous, they are often administered by end-users without any training in networking or security. As a result, many may have only minimal security safeguards. Further, some of these users may be budget-constrained and unlikely to purchase tools, such as enterprise middleboxes (MBes), that may be able to increase the security of their connected machines. Further, end-users may not carefully follow best security practices, such as updating device software and firmware regularly.

Residential networks may have high device diversity, including more traditional systems such as laptops or desktop computers, but also including Internet-enabled televisions, video game systems, and home automation systems. These embedded devices, sometimes called Internet of Things (IoT) devices, may have vulnerabilities that go unaddressed, either by the manufacturer or the end-user [23], [31].

Residential routers are in a strategic place to address the weaknesses of security in end-devices. Since the routers are key to communication between these devices and the untrusted Internet, the router can perform filtering and validation efforts to block attacks from exploiting devices. Unfortunately, residential routers are often resource constrained and lack the ability to enforce protections. However, when routers combine software-defined networking (SDN) techniques with cloud-based virtual machines (VMs), these routers can ensure that the middleboxes running on cloud VMs can enforce certain security goals [39].

While cloud-based middleboxes are useful in some circumstances, they may introduce unacceptable network latency or bandwidth overheads in some applications, such as online video games. Further, cloud providers must charge users for their network bandwidth, which incentivizes end-users and cloud-based security providers to minimize the traffic sent to cloud-based VMs.

Our goal in this work is to protect residential devices by ensuring the authenticity of the communication between the devices and outside systems. Essentially, if we can protect devices from communicating with untrustworthy third-party systems, we can prevent devices from being attacked. In several security protocols, such as TLS (the successor to SSL), SSH, and IPSec, the initial connection negotiation phase has the greatest vulnerability [13], since it requires confirmation of the other party's authenticity. Given the prominence of TLS in web security and online protocols, we focus on this protocol as a concrete example and later discuss how the approach can be applied to other protocols.

In this work, we ask three research questions: 1) To what extent can we perform in-line TLS certificate verification and revocation validation using cloud-based middleboxes? 2) How can we minimize the performance impact of cloud-proxying on long-lived network flows? 3) To what extent can SDN middleboxes provide novel support for other important security protocols?

In performing the work, we make the following contributions:

- **Implementation of a Novel Cloud-Based TLS Validator:** We created a new verifier, called *TLSDeputy*, that monitors the TLS handshake process and performs independent verification of TLS certificates and revocation checking using certificate revocation lists (CRLs). Such revocation checks were particularly important following the recent HeartBleed vulnerability [17].
- **Evaluation of the Cloud-Based Validator:** We verified the efficacy of the TLSDeputy across diverse devices and showed that it could increase device security. In particular, we showed that the

TLSDeputy prevented smartphone web browsers, which are known to not properly check for certificate revocations [29], from reaching untrustworthy web sites. Our approach behaves similar to a client performing full-chain TLS verification and revocation check and can feasibly be used today. Finally, we evaluated the tool across 40,000 top web sites and found that it properly determined which HTTPS servers were valid and which were not, demonstrating its real-world viability.

- **Created a Novel Communication Channel for the Middlebox:** By embracing the concept of participatory networks [19], we created a new communication channel between the OpenFlow SDN controller and the cloud-based middlebox. In doing so, we were able to migrate a network flow to use a direct path from the user's network after TLSDeputy confirms the TLS handshake was proper and authentic. This addresses known limitations of MB and controller consistency [18], [21].

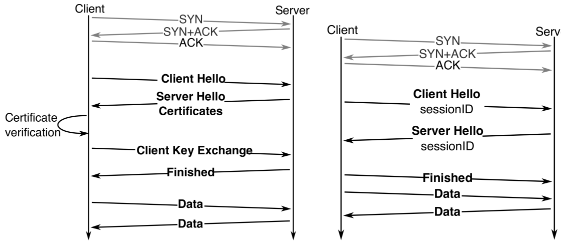
II. Background and Related Work

Given our emphasis on TLS as a working example, we provide a background on the protocol and on work that aims to improve the protocol. We then describe work for using SDNs to outsource residential network security.

A. TLS Background

All TLS connections are preceded by a TLS handshake in addition to a TCP handshake. Figure 1(a) shows a full TLS handshake[1] where the server provides the corresponding certificate chain. That certificate chain starts with a self-signed, well-trusted root certificate. The root certificate signs the next certificate in the chain, attesting to that certificate's validity. The process continues with each certificate signing the next one until the process concludes at the server's own certificate.

Upon receiving the certificates from the server, the client then verifies each certificate in the chain. After verification, the client and server create a session key to use for encrypting the data to be transmitted. As a performance enhancement, Figure 1(b) shows how future TLS connection establishment from the client can be abbreviated by transmitting a session ID that is cryptographically derived from a previous handshake. Since certificate verification happens early in the communication between the client and server, our approach can ignore the remaining TLS connection once the certificates are successfully verified.



(a) Full TLS Handshake (b) TLS Renegotiation

Fig. 1. TLS Handshake

B. TLS Research

Researchers have performed Internet-wide scans and those of the Alexa top 1 million [1] domain names in recent years. Holz *et al.* [25], [26] have performed multiple investigations of TLS certificates to determine characteristics such as error codes in verification, chain length, and ciphers. Zhang *et al.* [40] performed scans in response to the devastating Heartbleed [17] attack. This attack, and the subsequent analysis, shows the importance of a certificate revocation system. However, few end-hosts check for the actual revocation status of certificates a year after that attack. Liu *et al.* [29] found that, with the exception of Extended Validation (EV) certificates, there is wide-spread failure in desktop web browsers to check CRLs for certificate revocation lists and no mobile platform browsers did so.

TLS is vulnerable to man-in-the-middle (MITM) attacks when clients fail to properly verify certificates or when malware has installed new root certificates. Recent attacks have demonstrated the ease of deploying MITM attacks on some embedded devices [23]. Dacosta *et al.* [15] provides an efficient approach to detecting MITM attacks by allowing domain servers to use a previously-established, secure channel to provide additional information to directly vouch for certificates. Huang *et al.* [27] detects live MITM attacks by detecting forged certificates through a browser Flash application.

To prevent SSL attacks such as a MITM, researchers have used various methodologies for improving overall security. Georgiev *et al.* [22] found vulnerabilities in several security-critical applications and attributed the problem to application developers misinterpreting SSL library APIs. SSLint [24] was built as a static analysis tool that will detect applications that are misusing SSL APIs. Frankencerts [13] is a blackbox solution that automates the vulnerability detection process in SSL libraries by generating certificates to test for certain vulnerabilities. Our work is orthogonal in that TLSDeputy detects and prevents insecure connections.

Client resource and performance limitations have led to a number of research efforts. Server-Based Certificate Validation Protocol (SCVP) [20] is an approach

[1] Clients may also authenticate to the server but we exclude this case from our discussion.

to enabling clients to delegate path construction and certificate validation to another server. This proposed standard has similar goals to TLSDeputy but requires client-side support, which may not be feasible in legacy or embedded devices. Naylor *et al.* [30] broadly quantifies the performance costs associated with deploying HTTPS over HTTP, which includes additional latency and inability to effectively use caches. Zhu *et al.* [41] more specifically focuses on the performance associated with OCSP. While their work shows OCSP response times are getting better, Liu [29]'s work shows that CRLs are still the most popular revocation process for all certificates (leaf and intermediate CAs). For example, less that 50% of intermediate certificates support OCSP as compared to 99% that support CRLs.

C. Existing TLS Security Systems

Some browsers are taking steps to improve revocation checks. Chrome has introduced CRLSets [3] that contain an internally maintained list of CRLs. Which CRLs are included is not publicly known. However, the total size is limited to 250KB. Similarly, Firefox is beginning its own approach called OneCRL [8]. In contrast, our work actively maintains a large CRL database that does not need to compromise between CRL size and security.

ICSI Notary [4] is a system that passively collects certificates from participating gateways. Clients can perform DNS queries using a hashed digest of a certificate to the Notary. The DNS response contains information based about the certificate based on what the participating gateways have observed. The ICSI Notary's does not provide an enforcement mechanism but could provide another reference point for TLSDeputy's certificate validation.

Finally, Barracuda [2] has developed hardware to provide an inline application firewall that will maintain CRLs and perform OCSP checks for client certificates. That approach only focuses on revocation (no verification) and only for client certificates, which are rarely observed within the residential environment.

D. Outsourcing and Residential SDN

Our past work [39] used cloud-based servers to provide an enterprise-grade proxy solution. We motivated and evaluated our approach using the popular Skype VoIP application by automatically detecting and proxying Skype-related traffic through an anonymizing proxy. TLSDeputy builds upon this architecture and provides a detailed investigation of the performance characteristics of outsourcing security applications to a public cloud.

Other research has considered outsourcing residential network functionality [28], [38]. Unfortunately, these approaches require ISP support in deploying new services within the ISP's infrastructure. Such approaches typically require custom virtual gateways within the home, which do not exist in practice, and introduce new protocols for deploying functionality. While the ISP is well-positioned to enable such technologies, such support remains limited in practice. We avoid requiring support from the ISP by modifying existing commodity home routers to support OpenFlow and leveraging public cloud infrastructure.

APLOMB was an enterprise-focused solution to outsourcing network functionality to the cloud that required a specialized network gateway [36]. APLOMB further required DNS modifications to support redirection to cloud MBes. Our approach only requires software modifications to existing hardware in the home. Our work is tailored towards the residential network where for outgoing connections a loopback approach is necessary without proxying the connection.

III. Securing Connection Establishment

Our goal is to protect applications that conduct important security interactions at the beginning of a connection. As part of our running example, we show how our work supplements traditional TLS verification and provides a practical approach to enforce certificate revocation checks.

A. System Overview and Trusted Computing Base

Our system uses OpenFlow-enabled switches, cloud-based controllers and middleboxes, and custom OpenFlow agents (OFAs). In Figure 2, we show an overview of our system with logical OpenFlow protocol communication depicted using dotted lines.

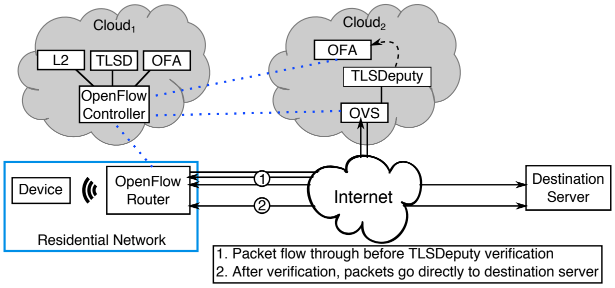


Fig. 2. Our system uses a cloud-based OpenFlow controller and middlebox for TLS verification and revocation. TLSDeputy relays verification results to a controller module using a special OpenFlow Agent (dashed line). Similarly, TLSDeputy has a module on the controller that steers new TLS connections through its MB software. Blue dotted lines represent logical communication using OpenFlow.

We consider all the cloud infrastructure, including the middlebox and OpenFlow controller, along with the residential OpenFlow router to be within our trusted computing base.

B. Cloud-based Flow Redirection

Our approach requires that some network traffic be inspected by MBes in the cloud. We use an OpenFlow controller and residential routers that support OpenFlow to redirect network flows as needed. Without a connection to an OpenFlow controller, our switch acts as a Layer 2 learning switch and mimics the behavior of traditional residential routers. That is, all required services for an Internet connection such as DHCP, DNS, and NAT all function without being connected to an OpenFlow controller. This allows us to safely fail-over in the event the OpenFlow controller goes offline. When connected, our OpenFlow router enforces fine-grained flow control. Any new network connection resulting in a new network 5-tuple (IP$_{src}$, IP$_{dst}$, Port$_{src}$, Port$_{dst}$, transport protocol) will require approval from the controller. The controller can then use packet-level information at the start of a connection to determine how the flow should be handled and whether or not a MB service is required.

By default, our controller performs basic Layer 2 learning to forward packets. In addition to Layer 2 learning, our controller runs a module to detect new TLS connections (labeled TLSD) and an OFA module that will communicate with the OpenFlow agent on the MB. When the TLSD module detects a new TLS connection, the TLSD module instructs the controller to send OpenFlow FlowMods to the Open vSwitch (OVS) instance in the cloud and to the home router. Those FlowMods will cause the router to tunnel all incoming and outgoing TLS packets through the cloud MB. These rules ensure that the MB will see the bidirectional communication between the client and TLS server.

The loopback communication path from the cloud MB, shown in Figure 2, allows us to remove the loop once the TLSDeputy has verified the TLS handshake. This restores the performance benefits of direct communication without the MB. If we instead proxied the connection through the MB, we would not need the loopback technique but would also never be able to migrate the connection away from the MB without breaking the end-to-end connection.

C. TLSDeputy Middlebox

Our TLSDeputy middlebox runs within a cloud VM that is connected to an OVS instance. The TLSDeputy monitors the TLS handshake and checks certificates and other important information, such as the Server Name Indicator (SNI) extensions to TLS, to ensure a secure TLS connection. In addition to checking for certificate revocation, TLSDeputy performs certificate verification and other similar tasks that the end-host also performs. We provide TLSDeputy with a trusted root certificate store containing 180 root certificates that were extracted from Mac OS X 10.11.3 to allow the TLSDeputy to verify certificate chains.

The OpenFlow controller detects TLS traffic using transport layer ports and diverts all TLS traffic to the TLSDeputy beginning with the TCP SYN packet. The TLSDeputy inspects the Client and Server Hello messages. First, the TLSDeputy checks to see if the TLS request is a renegotiation or a new connection. If both the client and server transmit a Session ID value in their handshakes, TLSDeputy recognizes the connection is a valid renegotiation and notifies the OpenFlow controller via the OFA that the communication can be transmitted directly via the residential router without further TLS-Deputy inspection. Otherwise, TLSDeputy knows the connection is a new negotiation and performs detailed verification checks.

If the client uses the SNI extension and specifies a server's host name, for example www.example.com, in the Client Hello message, we store that value to later verify the host name in the server's certificates. Next, the server responds with a Server Hello and immediately sends certificates, as shown in Figure 1(a). TLSDeputy parses the server's response and extracts each certificate being provided. As per RFC 5246 [16], the first certificate in the chain is the destination server's certificate. The subsequent certificates are then ordered such that the preceding certificate is directly certified by the next. The chain terminates, optionally, with the self-signed root certificate. Since TLSDeputy only trusts the root certificates that are pre-loaded in its local store, it ignores any self-signed root certificates.

Once the server sends the last certificate in the chain, TLSDeputy performs its verification before allowing the connection to continue. TLSDeputy passes the certificates and the client's indication of the server's host name, if any, to the verification submodule. For our verification, we use LibreSSL [5], which is a hardened implementation of the popular OpenSSL library. Since relatively few client implementations use LibreSSL currently, TLSDeputy's use of LibreSSL provides software diversity which may yield more robust security. We convert each certificate into a corresponding X509 certificate data structure and store each certificate. We use our root certificates to verify each of the provided certificates.

After completing the verification, TLSDeputy removes the flow from consideration and releases the remaining associated packets. TLS deputy can then watch the client's response to the packets. If a device proceeds with the connection when TLSDeputy found verification issues, TLSDeputy will detect the device is improperly verifying TLS handshakes and will break the connection. Optionally, the software can notify the user of the issue.

D. CRL Enforcement

Before the TLS certification chain can be verified, we must determine what CRL checks to perform. Due to implementation details in both LibreSSL and OpenSSL, there are only two options: only verify the server's certificate or verify the entire chain. If any certificate in the chain lacks a CRL, we cannot perform a full chain verification. Likewise, if the server's certificate lacks a CRL, no CRL verification is possible.

One of TLSDeputy's most important functions is to provide an approach that allow for efficient full path CRL enforcement. Recent work [29] has shown that no mobile browser performs revocation checks even after the high-profile Heartbleed attack. Liu *et al.* speculate that performance is likely a contributing factor given that their Internet-wide scan found the weighted average CRL size to be 51 KB. The size of CRL becomes more concerning as the length of the certificate chain grows. The average length of a valid chain has been shown to be 2 (a single intermediate CA) [11]. TLSDeputy addresses these concerns by proactively caching CRLs locally rather than obtaining them on demand.

To determine which CRL to consult, we check the CRL distribution point extension in each X509 object. For each certificate, we retrieve all the available the URIs distributions points[2] provided. Beginning with the server's certificate, we iteratively check for revocation using each certificate's indicated CRL. If we have successfully retrieved CRLs for all certificates in the chain, we perform a full-chain CRL check with LibreSSL.

E. Enforcing TLS Validation via Participatory OFAs

The TLSDeputy can be more efficient with assistance from the OpenFlow controller. If the TLSDeputy can communicate TLS verification information to the controller, the controller can then allow subsequent packets in the connection to be routed directly (if TLSDeputy verification passed) or install a drop rule at the residential router (if TLSDeputy verification failed). This optimization is an example of the "participatory networks" concept. Essentially, the OpenFlow controller enforces policy in the network yet relies upon MBes to perform detailed inspection that is not feasible at the controller. However, traditionally, the controller and MBes cannot share information and collaboratively enforce policy.

Others have attempted to address the problem of SDN and MBes by modifying packets in-flight to hold additional information. For example, FlowTags [18] overloads the 6-bit Differentiated Services field in the IP header of a packet to pass information between OpenFlow switches. OpenMB [21] suggests making the internal state of a MB accessible to the controller to allow the controller to understand what actions were taken. These approaches are limited in the amount of information they can share or in the amount of redesign necessary for support. To address this problem, we embrace the notion of participatory networking [19] whereby MBes can relay information to the controller to enable flow-level decisions. FRESCO [37] has a similar notion of enabling an API where MBes can send information out-of-band to their applications. In contrast, our approach, shown in detail in Figure 3, allows a MB to embed arbitrary information into an OpenFlow PacketIn message and transmit that in-band to the OpenFlow controller.

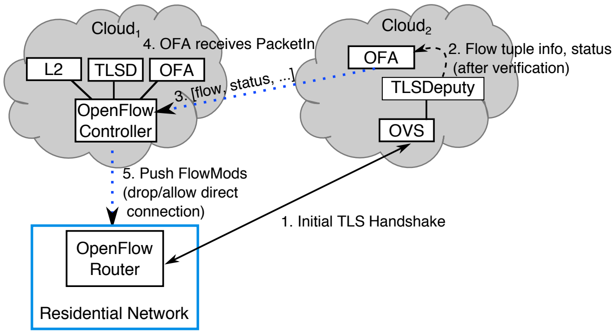


Fig. 3. Once the TLSDeputy has verified the handshake, it can contact the controller through a custom OpenFlow agent to request the connection be sent directly rather than diverted through the middlebox. The controller can then send FlowMods to the residential router causing packets to be transmitted directly rather than via a tunnel.

Although the middlebox communicates using an OpenFlow PacketIn event, the payload of that message uses a custom payload recognizable only by our own specific OpenFlow controller module. We configure the controller so that the only module listening for events from the OFA is the OFA module that we designed for this purpose. Accordingly, we can pass any arbitrary information to the module relating to MB state. In our work, we pass the flow tuple when verification has finished, the status (e.g., success or failure) and an additional message describing the reason for failure, if appropriate. Future work will integrate this approach with other MB applications such as an existing IDS.

F. Obtaining and Maintaining CRLs

Ideally, our approach maintains an Internet-wide cache of all CRLs. We move towards this goal by initially crawling the top 1 million Alexa domains [1] and obtaining the CRLs for each certificate in a given chain. The initial scan recovered 1,608 potentially reachable CRLs of which 1,495 were retrieved.

Our ideal goal is to maintain a complete list of all CRLs used on the Internet. As a result, anytime TLSDeputy encounters a certificate with a CRL not in the database we add the URI to a list of monitored CRLs and immediately begin retrieving it in the background. However, to avoid performance issues, we do not wait to check the CRL for the chain causing the first retrieval. Instead, we will enforce such revocation checks on the next connection that uses the CRL. For example, Apple's

[2] We ignore unreachable distribution points such as `ldap://` and `file://`.

Messages application regularly performs background TLS connections that have several CRLs that were not originally in our database. On the first connection, we will not be able to enforce the CRL, but we will be able to do so on the next connection. As we build our CRL database, we retrieve all CRLs every 12 hours, which is more frequent than the majority of the CRL validity lengths in the certificates we found.

IV. Implementation

To implement the TLSDeputy, we use custom router firmware on TP-LINK Archer C7 routers. We installed OpenWrt [10] and added the Open vSwitch [32] package for OpenFlow support. We used the POX [7] controller running on Amazon EC2 micro-instance VMs to manage the router. For tunneling, we used GRE tunnels as supported by OVS. This allowed better systematic tunneling control than our past work, which required a routing agent to direct flows over a Linux GRE tunnel [39]. When the controller detects a new TLS flow, the TLSD module uses these GRE tunnels for directing the TLS handshake through the TLDeputy middlebox.

Our TLSDeputy is a C++ application leveraging the LibreSSL [5] implementation for certificate verification. We implemented our own certificate stripping and parsing functionality. The CRL retrieval and maintenance code were written as scripts. We ran the TLSDeputy MB and controller in separate EC2 micro-instances.

Our OFA application is a custom OpenFlow 1.0 compliant agent that communicates over an OpenFlow connection to the controller and uses a local TCP socket to receive verification information from the TLSDeputy.

A. Managing MTU Restrictions

Since we are using built-in tunneling support from OVS, we must account for the overhead in bytes associated with GRE tunneling packets starting from Layer 2. The Maximum Transmission Unit (MTU) between networks is typically 1500 bytes. Without accounting for the GRE overhead, our packets could be dropped by intermediate routers before reaching the tunnel endpoint. One possibility for addressing this issue is to using IP fragmentation to split the packet and have it reassembled at the MB. IP fragmentation is typically avoided when ever possible due to performance concerns. Instead, we use the MB to set the Maximum Segment Size (MSS) to 1400 bytes in the TCP handshake of both the source and the destination. By reducing the MSS in the SYN/SYN+ACK packets, both end-points of the connection will reduce the payload of packets transmitted and thus avoid fragmentation altogether.

V. TLSDeputy Evaluation

We evaluate TLSDeputy's security effectiveness using two IoT devices, smartphone web browsers, and web browsers on traditional laptop/desktop operating systems. We then compare the performance of TLSDeputy against traditional certificate verification and revocation from a residential network.

A. Experimental Setup

For our security evaluation, we use multiple security testing software packages and our own certificate authority. Many IoT devices are hardcoded to communication with specific servers or domains. Accordingly, we use `mitmproxy` [14] and SSLsplit [34] to determine if these non-browser applications and devices properly verify TLS certificates and detect forgeries. We monitor network traffic from such devices to determine if the device performed revocation checking via OCSP or CRL retrievals. We created a self-signed root CA and a TLS chain consisting of a single intermediate certificate authority. Using the intermediate CA, we signed a leaf certificate for a publicly accessible web server. Our leaf certificate's revocation status was obtainable only via a CRL. After generating the web server's certificate, we immediately revoked it and updated the CRL accordingly. However, the web server was configured to continue using the revoked certificate.

The two IoT devices we use in testing are a Foscam IP camera, which is used for home surveillance, and a Belkin WeMo power outlet that can be turned on or off through a smartphone application.

B. Security Effectiveness

Our security evaluation focuses on IoT, mobile devices, and desktop browsers that operate within the home network. We compare how TLSDeputy operates in comparison to the software embedded on two IoT devices, both of which have known security vulnerabilities [6], [9]. We also perform tests using mobile devices using several major browser platforms. The results of security evaluation are shown in Table III-F.

We first describe the IoT device results. Unsurprisingly, neither the Focam or WeMo performed any type of revocation. WeMo has a reported verification vulnerability that a certificate store is not stored locally on the device. In our testing, we did not find that our device was vulnerable to MITM attacks. However, we did find the Foscam was vulnerable to such MITM attacks. Foscam's configuration allows users to setup notifications of motion detection with images through an email. During configuration, the user must provide a mail server configuration, including a domain name and port, and if authentication is required, a username and password as well. Mail servers such as Gmail require a TLS connection for sending and receiving email. Our research found that the Foscam is indeed vulnerable to a MITM attack on the communication between the camera

TABLE I

EVALUATION OF TLSDEPUTY ON IOT AND MOBILE PLATFORMS WITH A REVOKED LEAF CERTIFICATE

Device Type	Device		Device Verification	TLSDeputy Verification	Device Revocation	TLSDeputy Revocation
IoT	Foscam		✗	✓	✗	✓
	WeMo		✓	✓	✗	✓
Mobile	iPhone	Safari	✓	✓	✗	✓
		Chrome	✓	✓	✗	✓
		Firefox	✓	✓	✗	✓
	Android	Default	✓	✓	✗	✓
		Chrome	✓	✓	✗	✓
		Firefox	✓	✓	✗	✓
Desktop	Mac OS X	Safari	✓	✓	✗	✓
		Chrome	✓	✓	✗	✓
		Firefox	✓	✓	✗	✓
	Linux	Chrome	✓	✓	✗	✓
		Firefox	✓	✓	✗	✓
	Windows	IE	✓	✓	✓	✓
		Chrome	✓	✓	✓	✓
		Firefox	✓	✓	✗	✓

and the Gmail mail servers, which can expose a user's Gmail username and password. We found none of the listed CVE's for Foscam [6] discuss TLS vulnerabilities and conclude this was previously undocumented. Fortunately, our TLSDeputy system is able to detect and block this MITM attack without requiring software updates to the Foscam or support from the manufacturer[3]. Without TLSDeputy, it would be very difficult to determine if a MITM attack was occurring on any IoT device.

During our evaluation, we expected that mobile browsers would perform proper verification. Indeed, without installing our root certificate on the mobile device, all browsers detected the certificate was untrusted, stopped the connection, and notified the user. After these tests, we installed our root certificate on all each device in order to have the browsers trust the certificate chain and then attempt a new TLS connection. After establishing the connection, none of the mobile browsers we tested performed revocation checks on our server's certificate, which corroborates recent research [29]. In contrast to Liu's work, we found that the newest version of Safari (v9) did not properly check our CRL for revocation. Their tests covered through v8. Additionally, we found that Chrome v49 did properly check the revocation status. Liu *et al.* [29] found that Chrome v44 only performed this check for Extended Validation (EV) certificates, which our certificate was not. Chrome may have recently updated its revocation process. Again, TLSDeputy uses its cached CRL to block connections for each browser as shown in Table III-F, protecting even devices and applications that fail to perform the appropriate verification or revocation checks.

C. Performance Results

Our performance experiments present two different comparisons. We first consider the end-to-end performance of using TLSDeputy versus traditional end-host verification when only considering the leaf certificate for revocation. Our other performance experiment compares TLSDeputy to full path revocation checks using CRLs. The results were obtained from a residential cable network in Massachusetts with Amazon EC2 instances hosted in the North Virginia data center.

1) TLS Verification and Revocation Overhead: Virtually no desktop or mobile browser performs full chain verification using CRL or OCSP. Given the frequency of browsers only checking leaf certificates, we perform head-to-head performance measurements over 40,000 random domains using OCSP and CRLs to performing revocation checks on leaf certificates. We then performed connections using TLSDeputy to the same domains.

For OCSP and CRL leaf certificate revocation checking, we first performed a TLS handshake with only verification (i.e., not checking for revocation). Upon verification, we obtained the leaf certificate's OCSP URL and each of the certificates provided during the handshake to check the leaf certificate's revocation status. We then added the time to perform the OCSP check to the TLS handshake time. Similarly, we obtained the CRL distribution point from the TLS handshake and performed a file retrieval on the CRL. The time taken to retrieve the CRL file was added to the base TLS handshake time. Lastly, we initiated a new TLS handshake with TLSDeputy enabled, but allowed TLSDeputy to also perform revocation checks on intermediate certificates. The results are presented in Figure 4 and shows that TLSDeputy adds roughly 0.5 seconds to the median of an TLS handshake.

2) Full Chain Revocation Overhead: Only 48.5% of intermediate certificates (which excludes leaf certificates CRL) offer OCSP for revocation checking [29]. This low number of OCSP responders means that the majority of full path revocation checks require CRLs. To better understand the impacts of full chain revocation checks, we perform an additional experiment using 10,000 random domains which have two or more CRLs in the chain.

[3] Prior to publishing this work, we contacted both manufacturers and disclosed these vulnerabilities and suggested remediation approaches.

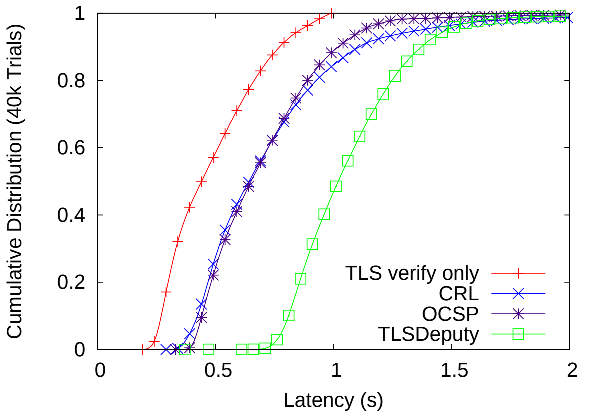


Fig. 4. Leaf certificate verification comparison between CRL, OCSP, and TLSDeputy over 40k random domains.

Similar to our previous experiment, we first initiate a TLS handshake and then retrieve each CRL in the path while accumulating the total time for the connection and each CRL retrieval. The results of this experiment on shown in Figure 5 and show the overhead associated with full chain revocation checks using CRLs is comparable to TLSDeputy's performance.

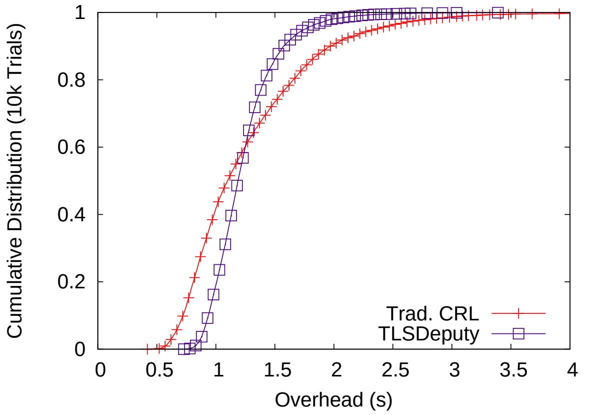


Fig. 5. Complete chain verification using CRLs.

3) Viability in Practice: In performing the verification across 40,000 domains, we found that TLSDeputy was viable in practice and was able to determine which TLS connections were valid and which were not.

D. Evaluation Summary

Our approach is able to protect vulnerable devices, including IoT devices, from connecting to servers with invalid certificates. Further, we are able to protect many IoT, mobile, and desktop devices that do not properly check for certificate revocation. The performance costs for doing so are comparable to a full chain CRL verification at the client. Essentially, our middlebox strategy is able to provide whole network protections for a residential network at roughly the same cost of doing the appropriate verifications at each end device.

VI. Discussion

While we have focused on TLS in this paper, the same approach is viable for other security protocols such as SSH and IPSec. In particular, SSH's leap-of-faith security approach, in which a user may accept a public key for a server without verifying it, has recognized security risks [12]. We can eliminate the need for a leap-of-faith by combining the use of DNSSEC and the `SSHFP` resource record [35]. Our middlebox could intercept DNS responses, cryptographically verify the `SSHFP` records using DNSSEC, and store the destination IP address and SSH fingerprint for each server in a temporary database. For any SSH connections to known IP addresses, the middlebox would then verify the public key matched. With our tool, an organization could configure DNSSEC and `SSHFP` records to ensure any clients using our approach would be protected from SSH man-in-the-middle attacks during the first SSH connection.

We can protect IPSec authenticity in a manner similar to SSH. Using DNSSEC and the `KEY` resource record [33], the middlebox can perform the appropriate verification to ensure the IPSec server's authenticity.

From a cost perspective, our development and evaluation cost approximately $20 per month for cloud hosting. The costs included two always-on VMs, network traffic transmission, and disk storage, with the majority of the cost associated with the VM uptime. Given our minimal CPU and memory overheads, multiple residential networks could easily share these VMs. Practically, a third-party security provider could run cloud-based VMs to provide TLSDeputy services to large numbers of residential users and achieve economies of scale.

VII. Conclusion

In this work, we present TLSDeputy, a system that allows residential networks to ensure they only connect to properly verified TLS servers. We have shown the approach offers valuable security protections for IoT, mobile, and desktop devices and that the performance is comparable to correct client-side verification measures. Finally, using a set of 40,000 servers, we have demonstrated the approach is capable of verifying connections to top TLS destinations and can immediately be deployed to residential networks.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant No. 1422180. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

[1] Alexa top 1 million domains. http://s3.amazonaws.com/alexa-static/top-1m.csv.zip. (Accessed on 04/05/2016).

[2] Barracuda web application firewall - client certificate validation using OCSP and CRLs. https://techlib.barracuda.com/waf/clientcertvalidation. (Accessed on 04/18/2016).

[3] CRLSets - the chromium projects. https://dev.chromium.org/Home/chromium-security/crlsets. (Accessed on 04/18/2016).

[4] The ICSI certificate notary. https://notary.icsi.berkeley.edu/#how-it-works. (Accessed on 04/18/2016).

[5] LibreSSL. http://www.libressl.org/. (Accessed on 04/15/2016).

[6] MITRE CVE - search results. https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=foscam. (Accessed on 04/05/2016).

[7] POX. http://www.noxrepo.org/pox/about-pox.

[8] Revoking intermediate certificates: Introducing OneCRL. https://blog.mozilla.org/security/2015/03/03/revoking-intermediate-certificates-introducing-onecrl/. (Accessed on 04/18/2016).

[9] Vulnerability note VU#656302 - Belkin Wemo home automation devices contain multiple vulnerabilities. https://www.kb.cert.org/vuls/id/656302. (Accessed on 04/05/2016).

[10] OpenWrt wireless freedom. https://openwrt.org, 2014.

[11] B. Amann, M. Vallentin, S. Hall, and R. Sommer. Revisiting SSL: A large-scale study of the Internet's most trusted protocol. Technical report, Citeseer, 2012.

[12] J. Arkko and P. Nikander. Weak authentication: How to authenticate unknown principals without trusted parties. In *Security Protocols*, pages 5–19. Springer, 2002.

[13] C. Brubaker, S. Jana, B. Ray, S. Khurshid, and V. Shmatikov. Using Frankencerts for automated adversarial testing of certificate validation in SSL/TLS implementations. In *IEEE Symposium on Security and Privacy*, SP '14, pages 114–129, Washington, DC, USA, 2014. IEEE Computer Society.

[14] A. Cortesi. mitmproxy. https://mitmproxy.org/. (Accessed on 04/05/2016).

[15] I. Dacosta, M. Ahamad, and P. Traynor. Trust no one else: Detecting MITM attacks against SSL/TLS without third-parties. In *Computer Security–ESORICS 2012*, pages 199–216. Springer, 2012.

[16] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard), Aug. 2008. Updated by RFCs 5746, 5878, 6176, 7465.

[17] W. Dormann. Vulnerability note VU#720951 - OpenSSL TLS heartbeat extension read overflow discloses sensitive information. https://www.kb.cert.org/vuls/id/720951. (Accessed on 03-05-2016).

[18] S. K. Fayazbakhsh, L. Chiang, V. Sekar, M. Yu, and J. C. Mogul. Enforcing network-wide policies in the presence of dynamic middlebox actions using flowtags. In *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*, NSDI'14, pages 533–546, Berkeley, CA, USA, 2014. USENIX Association.

[19] A. D. Ferguson, A. Guha, C. Liang, R. Fonseca, and S. Krishnamurthi. Participatory networking: An API for application control of SDNs. In *ACM SIGCOMM Computer Communication Review*, volume 43, pages 327–338. ACM, 2013.

[20] T. Freeman, R. Housley, A. Malpani, D. Cooper, and W. Polk. Server-Based Certificate Validation Protocol (SCVP). RFC 5055 (Proposed Standard), Dec. 2007.

[21] A. Gember, R. Grandl, J. Khalid, and A. Akella. Design and implementation of a framework for software-defined middlebox networking. *SIGCOMM Comput. Commun. Rev.*, 43(4):467–468, Aug. 2013.

[22] M. Georgiev, S. Iyengar, S. Jana, R. Anubhai, D. Boneh, and V. Shmatikov. The most dangerous code in the world: Validating SSL certificates in non-browser software. In *ACM Conference on Computer and Communications Security*, CCS '12, New York, NY, USA, 2012. ACM.

[23] D. Goodin. Man-in-the-middle attack on Vizio TVs coughs up owners' viewing habits. http://arstechnica.com/security/2015/11/man-in-the-middle-attack-on-vizio-tvs-coughs-up-owners-viewing-habits/. (Accessed 03-05-2016).

[24] B. He, V. Rastogi, Y. Cao, Y. Chen, V. Venkatakrishnan, R. Yang, and Z. Zhang. Vetting SSL usage in applications with SSLint. In *Security and Privacy (SP)*, pages 519–534. IEEE, 2015.

[25] R. Holz, J. Amann, O. Mehani, M. Wachs, and M. A. Kaafar. TLS in the wild: an Internet-wide analysis of TLS-based protocols for electronic communication. *arXiv preprint arXiv:1511.00341*, 2015.

[26] R. Holz, L. Braun, N. Kammenhuber, and G. Carle. The SSL landscape: A thorough analysis of the x.509 PKI using active and passive measurements. In *Internet Measurement Conference*, pages 427–444, New York, NY, USA, 2011. ACM.

[27] L. S. Huang, A. Rice, E. Ellingsen, and C. Jackson. Analyzing forged SSL certificates in the wild. In *Security and Privacy (SP)*, pages 83–97. IEEE, 2014.

[28] K. R. Khan, Z. Ahmed, S. Ahmed, A. Syed, and S. A. Khayam. Rapid and scalable ISP service delivery through a programmable middlebox. *SIGCOMM Computer Communication Review*, 44(3):31–37, July 2014.

[29] Y. Liu, W. Tome, L. Zhang, D. Choffnes, D. Levin, B. Maggs, A. Mislove, A. Schulman, and C. Wilson. An end-to-end measurement of certificate revocation in the web's PKI. In *Internet Measurement Conference*, pages 183–196. ACM, 2015.

[30] D. Naylor, A. Finamore, I. Leontiadis, Y. Grunenberger, M. Mellia, M. Munafò, K. Papagiannaki, and P. Steenkiste. The cost of the S in HTTPS. In *ACM Conference on Emerging Networking Experiments and Technologies*, pages 133–140. ACM, 2014.

[31] C. Neagle. Smart refrigerator hack exposes Gmail account credentials. http://www.networkworld.com/article/2976270/internet-of-things/smart-refrigerator-hack-exposes-gmail-login-credentials.html. (Accessed on 03-05-2016).

[32] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, et al. The design and implementation of Open vSwitch. In *Networked Systems Design and Implementation (NSDI)*, pages 117–130, 2015.

[33] M. Richardson and D. H. Redelmeier. Opportunistic encryption using the Internet key exchange (IKE). IETF RFC 4322, December 2005.

[34] D. Roethlisberger. SSLsplit - transparent SSL/TLS interception). https://www.roe.ch/SSLsplit. (Accessed on 04/05/2016).

[35] J. Schlyter and W. Griffin. Using DNS to securely publish secure shell (SSH) key fingerprints. IETF RFC 4255, January 2006.

[36] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar. Making middleboxes someone else's problem: network processing as a cloud service. *ACM SIGCOMM Computer Communication Review*, 42(4):13–24, 2012.

[37] S. Shin, P. A. Porras, V. Yegneswaran, M. W. Fong, G. Gu, and M. Tyson. Fresco: Modular composable security services for software-defined networks. In *NDSS*, 2013.

[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 *Wireless and Mobile Computing, Networking and Communications (WiMob)*, pages 163–167. IEEE, Oct 2015.

[39] C. R. Taylor, C. A. Shue, and M. E. Najd. Whole home proxies: Bringing enterprise-grade security to residential networks. In *IEEE International Conference on Communications (ICC)*, May 2016.

[40] L. Zhang, D. Choffnes, D. Levin, T. Dumitras, A. Mislove, A. Schulman, and C. Wilson. Analysis of ssl certificate reissues and revocations in the wake of heartbleed. In *Internet Measurement Conference*, pages 489–502. ACM, 2014.

[41] L. Zhu, J. Amann, and J. Heidemann. Measuring the latency and pervasiveness of tls certificate revocation. In *Passive and Active Measurement*, pages 16–29. Springer, 2016.