The Sorry State of TLS Security in Enterprise Interception Appliances[*]

LOUIS WAKED, MOHAMMAD MANNAN, and AMR YOUSSEF, Concordia Institute for Information Systems Engineering, Concordia University, Montreal, Canada

Network traffic inspection, including TLS traffic, in enterprise environments is widely practiced. Reasons for doing so are primarily related to improving enterprise security (e.g., phishing and malicious traffic detection) and meeting legal requirements (e.g., preventing unauthorized data leakage and copyright violations). To analyze TLS-encrypted data, network appliances implement a Man-in-the-Middle TLS proxy, by acting as the intended web server to a requesting client (e.g., a browser), and acting as the client to the actual/outside web server. As such, the TLS proxy must implement both a TLS client and a server, and handle a large amount of traffic, preferably, in real-time. However, as protocol and implementation layer vulnerabilities in TLS/HTTPS are quite frequent, these proxies must be, at least, as secure as a modern, up-to-date web browser, and a properly configured web server (e.g., an A+ rating in SSLlabs.com). As opposed to client-end TLS proxies (e.g., as in several anti-virus products), the proxies in network appliances may serve hundreds to thousands of clients, and *any* vulnerability in their TLS implementations can significantly downgrade enterprise security.

To analyze TLS security of network appliances, we develop a comprehensive framework, combining and extending tests from existing work on client-end and network-based interception studies. We analyze thirteen representative network appliances over a period of more than a year (including versions before and after notifying affected vendors, a total of 17 versions), and uncover several security issues. For instance, we found that four appliances perform no certificate validation at all, three use pre-generated certificates, and eleven accept certificates signed using MD5, exposing their clients to MITM attacks. Our goal is to highlight the risks introduced by widely-used TLS proxies in enterprise and government environments, potentially affecting many systems hosting security, privacy, and financially sensitive data.

CCS Concepts: • **Networks → Middle boxes / network appliances**; • **Security and privacy → Network security**; Browser security.

Additional Key Words and Phrases: TLS Interception, middleboxes, network appliances, man-in-the-middle, server impersonation

ACM Reference Format:
Louis Waked, Mohammad Mannan, and Amr Youssef. 2019. The Sorry State of TLS Security in Enterprise Interception Appliances. *Digit. Threat. Res. Pract.* 1, 1 (July 2019), 26 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

[*]This work is the extension of an ACM ASIACCS 2018 paper [57].

Authors' address: Louis Waked, louiswaked@gmail.com; Mohammad Mannan, mmannan@ciise.concordia.ca; Amr Youssef, youssef@ciise.concordia.ca, Concordia Institute for Information Systems Engineering, Concordia University, Montreal, Canada.

1 INTRODUCTION

A network appliance, typically deployed inline at the edge of networks, is a system that is either provided as a plug and play physical box from the vendor, a virtual image or a cloud service. Such appliances provide numerous features, which could allow it to act as a router, firewall, IDS/IPS, proxy, content filterer, load balancer, VPN server, DHCP/DNS server, and/or authentication server.

Most network appliances currently include an SSL/TLS interception feature in their products. The interception process is performed by making use of a TLS web proxy server. Being either transparent or explicit to the end-user, the proxy intercepts the user's request to visit a TLS server, and creates two separate TLS connections. It acts as the HTTPS endpoint for the user's browser, and as the client for the actual HTTPS web server. Having the appropriate private key for the signing certificate (inserted to the client's root CA store), the proxy has access to the raw plaintext traffic, and can perform any desired action, such as restricting the access to the web page by parsing its content, or passing it to an anti-virus/malware analysis module or a customized traffic monitoring tool. Common reasons for adopting TLS interception include the protection of organization and individuals against malware and phishing attacks, law enforcement and surveillance, access control and web filtering, national security, hacking and spying, and privacy and identity theft [51].

While interception violates the implicit end-to-end guarantee of TLS, we focus on the potential vulnerabilities that such feature introduces to end-users located behind the network appliances, following several other existing studies on TLS interception, e.g., [8, 21, 39, 42, 48]. In general, TLS interception, even if implemented correctly, still increases the attack surface on TLS due to the introduction of an additional TLS client and server at the proxy. However, the lack of consideration for following the current best practices on TLS security as implemented in modern browsers and TLS servers, may result in severe potential vulnerabilities, and overall, a significantly weak TLS connection. For example, the proxy may not mirror the TLS version and certificate parameters or might accept outdated, insecure ones. Also, the proxy could allow TLS compression, enabling the CRIME attack [41], or insecure renegotiation [50]. The proxy may downgrade the Extended Validation (EV) domains to Domain Validated (DV) ones. It also may not mirror the cipher suites offered by the requesting client, and use a hard-coded list with weak and insecure ciphers, reviving old attacks such as FREAK [34], Logjam [32], and BEAST [40]. If the proxy does not implement a proper certificate validation mechanism, invalid and tampered certificates could be accepted by the proxy, and the clients (as they see only proxy-issued, valid certificates). Accepting its own root certificate as the signing authority of externally delivered content could allow MITM attacks on the network appliance itself. The use of a pre-generated key pair, which is bundled with an installer and shared among all installed copies, by a proxy could enable a generic trivial MITM attack [39]. In addition, the proxy may rely on an outdated root CA store for certificate validation, containing certificates with insecure key length, expired certificates, or banned certificates that are no longer trusted by major browsers/OS vendors.

We argue that most past studies on network appliances analyzed mostly preliminary and/or network measurement aspects of TLS interception (see Section 3), while the extensive work of Carnavalet and Mannan [39] targeted only client-end TLS proxies. However, TLS vulnerabilities in network appliances could result in more serious security issues, as arguably, enterprise computers handle more important business/government data in bulk, compared to personal information on a home user machine. Also, a single, flawed enterprise TLS proxy can affect hundreds of business users, as opposed to one or few users using a home computer with a client-side TLS proxy.

We present an extensive framework dedicated for analyzing TLS intercepting appliances, borrowing/adapting several aspects of existing work on network appliances and client-end proxies, in addition to applying a set of comprehensive certificate validation tests. We analyze the TLS-related behaviors of appliance-based proxies, and their potential vulnerabilities from several perspectives: TLS version and certificate parameter mapping, cipher suites, private key generation/protection, content of root CA store, known TLS attacks, and 32 certificate validation tests. We use this framework to evaluate 13 representative TLS network appliances, a total of 17 product versions, between July 2017 and March 2018 (see Table 1), including open source, free, low-end, and high-end network appliances, and present the vulnerabilities and flaws found. All our findings have been disclosed to the respective companies.

Analyzing network appliances raises several new challenges compared to testing browsers and client-end TLS proxies. Several network appliances do not include an interface for importing custom certificates (essential for testing), and many appliances do not provide access to the file system or a terminal, overburdening the tasks of injecting custom certificates and locating the private keys (for details, see Appendix B). Many appliances do not support more than one or two network interfaces, and thus, require the use of a router to connect to multiple interfaces. In addition, appliances that perform SSL certificate caching require the generation of a new root key pair for their TLS proxies for each test.

Our contributions can be summarized as follows.

- We develop a comprehensive framework to analyze TLS interception in enterprise-grade network appliances, combining our own certificate validation tests with existing tests for TLS proxies (both client-end services and network appliances), which we reuse or adapt as necessary for our purpose. Our certificate validation tests can be found at: https://madiba.encs.concordia.ca/software/tls-netapp/.
- We use this framework to evaluate thirteen well-known appliances from all tiers: open source, free, low-end, and high-end products, indicating that the proposed framework can be applied to different types of network appliances.
- We uncover several vulnerabilities and bad practices in the analyzed appliances, including: either an incomplete or completely absent certificate validation process (resulting trivial MITM attacks), improper use of TLS parameters that mislead clients, inadequate private key protection, and the use of weak/insecure cipher suites.
- Our specific findings that can easily enable MITM attacks include the following. Three appliances use pre-generated key pairs; one accepts self-signed certificates; four do not perform *any* certificate validation by default, and one omits validation even after explicitly enabling the feature in its configuration; eleven appliances accept certificates signed using MD5, and four appliances accept certificates signed using MD4; eight appliances offer weak and insecure ciphers and four support SSL 3.0, of which one only accepts TLS 1.0 and SSL 3.0; all appliances' root CA stores include at least one or more certificates deemed untrusted by major browser/OS vendors, and one includes an RSA-512 certificate, which can be trivially compromised.

Differences with the ACM ASIACCS version [57]. We make the following significant changes to the current article. We analyze seven new appliances using our framework (in addition to the six appliances in [57]); the updated results are discussed in Section 5. In total, we report the results of seventeen different versions of thirteen products we analyzed. We uncover a new dangerous vulnerability that was not seen with the previously tested appliances; three of the newly tested appliances use pre-generated root key pairs. This can lead to trivial full server impersonation attacks; we discuss the implications of this vulnerability in Section 6. We also reanalyze the latest up-to-date releases of the six previously analyzed appliances, present the new results in Section 5, and discuss the differences between the findings of the previously tested releases and the new ones in Section 7. Note that we shared our findings with the affected product vendors before our initial publication, and tested the versions that some vendors claimed to have fixed the reported vulnerabilities.

Table 1. List of the tested appliances; we use the text in bold to refer to the appliances throughout this paper. We only discuss the results of the latest releases ("Version" in bold).

Category	Appliance	Company	Version
	pfSense	NetGate	2.3.4
Linux			**2.4.2-P1**
Open-Source	**OpenSense**	Deciso B.V.	**18.1.2_2**
	Endian Firewall Community	Endian	**3.2.4**
	Untangle NG Firewall	Untangle	13.0
			13.2
	WebTitan Gateway	TitanHQ	5.15 build 794
			5.16 build 1602
	UserGate Web Filter	Entensys	**4.4.3320601**
Linux	**Cisco** Ironport WSA	Cisco	AsyncOS 10.5.1 build 270
Commercial			**AsyncOS 10.5.1 build 296**
	Sophos UTM	Sophos	**9.506-2**
	TrendMicro Interscan WSVA	TrendMicro	**6.5 SP2 build 1765**
	McAfee Web Gateway	McAfee	**7.7.2.8.0 build 25114**
	Cacheguard Web Gateway v1.3.5	Cacheguard	**1.3.5**
	Comodo Dome Firewall	Comodo	**2.3.0**
Windows	**Microsoft** TMG	Microsoft	**2010 SP2 rollup update 5**

2 BACKGROUND

In this section, we describe the TLS interception process, list the tested products, state the expected behavior of a TLS proxy, and explain the threat model.

Terminology. Throughout the paper, we refer to the TLS intercepting network appliances as proxies, HTTPS proxies, TLS proxies, middleboxes, or simply appliances. For the TLS requesting client, we use: browser, end-user, user, or client. The term *mirroring* is used to describe a situation where the proxy sends the same TLS parameters received from the web server to the client side, and vice versa; otherwise, *mapping* is used to indicate that the proxy has modified some parameters (for better or worse). We refer to the trusted root CA stores as stores, trusted stores or trusted CA stores. We refer to virtual machines as virtual appliances, VMs, or simply machines.

2.1 Proxies and TLS Interception

For TLS interception, network appliances make use of TLS proxies, deployed as either transparent proxies or explicit proxies. The explicit proxy requires the client machine or browser to have the proxy's IP address and listening port specifically configured to operate. Thus, the client is aware of the interception process, as the requests are sent to the proxy's socket. On the other hand, transparent proxies may operate without the explicit awareness of the clients, as they intercept outgoing requests that are meant for the web servers, without the use of an explicit proxy configuration on the client side; however, for TLS interception, a proxy's certificate must be added to the client's trusted root CA store (explicitly by the end-user, or pre-configured by an administrator). Such proxies could filter all ports, or a specific set of ports, typically including HTTP port 80 and HTTPS port 443. Secure email protocols could also be intercepted, by filtering port 465 for secure SMTP, port 993 for secure IMAP, and port 995 for secure POP3. The proxy handles the client's outgoing request by acting as the TLS connection's endpoint, and simultaneously initiates a new TLS connection to the actual web server by acting as the client, while relaying the two connections' requests and responses.

By design, the TLS protocol should prevent any MITM interception attempt, by enforcing a certificate validation process, which mandates that the incoming server certificate must be signed by a trusted issuer. Certificate authorities only provide server certificates to validated domains, and not to forwarding proxies, precluding the proxy from becoming a trusted issuer (i.e., a *valid* local CA). To bypass this restriction, the proxy can use a self-signed certificate that is added to the trusted root CA store of the TLS client, and thereby allowing the proxy to sign certificates for *any* domain on-the-fly, and avoid triggering browser warnings that may expose the untrusted status of the proxy's certificate. Thereafter, all HTTPS pages at the client will be protected by the proxy's certificate, instead of the intended external web server's certificate. Users are not usually aware of the interception process, unless they manually check the server certificate's issuer chain, and notice that the issuer is a local CA [49].

2.2 Tested Appliances

Most current network appliance vendors offer products for TLS interception. We select thirteen products, including: free appliances, appliances typically deployed by small companies, appliances with affordable licensing for small to medium sized businesses, and high-end products for large enterprises. We categorize the tested appliances into the following: Linux/Open-Source, Linux/Commercial and Windows; see Table 1.

For all the analyzed appliances, we keep the default configuration for their respective TLS proxies. An administrator could of course manually modify this default configuration, which may improve or degrade the proxy's TLS security. We thus choose to apply our test framework on the unmodified configuration (assuming the vendors will use *secure-defaults*).

2.3 Expected Behavior of a TLS Proxy

We summarize expected behaviors from a prudent interception proxy (following [39]). Deviations from these behaviors help design and refine our framework and validation tests.

The TLS version, key length, and signature algorithms should be mirrored (between client-proxy and proxy-web) to avoid misleading clients regarding the TLS security parameters used in the proxy-web connection. The client's cipher suites also should be mirrored, or at least the proxy must not offer any weak ciphers. The proxy must be patched against known TLS attacks, e.g., BEAST [40], CRIME [41], FREAK [34], Logjam [32], and TLS insecure renegotiation [50]. For more information on these attacks, see Appendix C.

TLS proxies must properly validate external certificates, as the client software (e.g., browsers) will be only exposed to the proxy-issued certificates. Thus, all aspects of TLS chain of trust should be properly validated, and common flaws must be avoided, e.g., untrusted issuers, mismatched signatures, wrong common-names, constrained issuers, revoked/expired certificates, and deprecated signature algorithms. The proxy's trusted CA store must avoid short key, expired or untrusted issuer certificates. TLS proxies must also disallow any external certificates signed by their own root keys. Proxies' private keys must be adequately protected (e.g., limiting access to the root account), and the keys must not be pre-generated (cf. [13]).

2.4 Threat Model

We mainly consider three types of attackers.

An *external attacker* can impersonate any web server by performing a MITM attack on a network appliance that does not perform a proper certificate validation. The attacker could be anywhere on the network between the appliance and the target website. Even if the validation process is perfect, the attacker could still impersonate any web server, if the appliance uses a pre-generated root certificate or accepts external site-certificates signed by its own root key. The attacker could also take advantage of known TLS attacks/vulnerabilities to potentially acquire authentication cookies (BEAST, CRIME), or impersonate web servers (FREAK, Logjam).

A *local attacker* (e.g., a malicious insider, external attacker with access to a compromised machine in the local network) with a network sniffer in promiscuous mode can get access to the raw traffic from the connections between the network appliance and clients. If the appliance uses a pre-generated certificate, the malicious user can install her own instance of the appliance, acquire its private key, and use it to decrypt the sniffed local traffic when the TLS connections are not protected by forward-secure ciphers. Such an adversary can also impersonate the proxy itself to other client machines, although this may be easily discovered by network administrators.

An *attacker who compromises the network appliance* itself with non-root privileges can acquire the private key if the key is not properly protected (e.g., read access to 'other' users and no passphrase encryption). With elevated privileges, more powerful attacks can be performed (e.g., beyond accessing/modifying TLS traffic). We do not consider such privileged attackers, assuming having root access on the appliance would be much more difficult than compromising other low-privileged accounts. Note that, in most cases, the *appliance* is simply an ordinary Linux/Windows box with specialized software/kernel, resulting in a large trusted code base (TCB).

3 RELATED WORK

Several studies have been recently conducted on TLS interception, TLS certificate validation, and forged TLS certificates. We briefly review studies that are closely related to our work.

Interception. Jarmoc [48] uncovered several TLS vulnerabilities in the certificate validation process of four network appliances using a test framework with seven certificate validation checks. Dormann [8, 21] relied on badssl.com's tests to check for vulnerabilities in two network appliances, finding flaws in the certificate validation process and the acceptance of insecure TLS parameters. Dormann also compiled a list of possibly affected software and hardware appliances.

Carnavalet and Mannan [39] proposed an extensive framework for analyzing client-end TLS intercepting applications, such as anti-virus and parental control software. They analyzed 14 applications (under Windows 7), revealing major flaws such as pre-generated certificates, faulty certificate validation, insecure private key protection, improper TLS parameter mapping, vulnerabilities to known TLS attacks, and unsanitized trusted CA stores. Durumeric et al. [42] later additionally included 5 TLS proxies under Mac OS, and 12 network appliances. They found that TLS proxies under Mac OS introduce more flaws than their Windows counterparts. They also showed that web servers can detect TLS interception, through the HTTP User-Agent header and protocol fingerprinting.

In March 2017, US-CERT [28] published an alert regarding TLS interception, to raise awareness of the dangers of TLS interception and its impact. Ruoti et al. [51] surveyed 1976 individuals regarding TLS inspection, to understand user opinion regarding legitimate uses of TLS inspection. Over 60% of the surveyed individuals had a negative response towards TLS inspection, and cited malicious hackers and governments as their main concerns.

Certificates scans. Huang et al. [47] analyzed over three million real-world TLS connections to facebook.com to detect forged certificates. They found that around 0.2% of the analyzed connections make use of a forged certificate, caused mainly by anti-virus software, network appliances and malware. O'Neill et al. [49] analyzed over 15 million real-world TLS connections using Google AdWords campaigns. They found that nearly 0.4% of the TLS connections were intercepted by TLS proxies, mostly by anti-virus products and network appliances, with the highest interception rates in France and Romania. In addition, Issuer Organization fields in some certificates matched the names of malware, such as 'Sendori, Inc', 'Web-MakerPlus Ltd', and 'IopFailZeroAccessCreate'.

Certificates validation. Fahl et al. [43] analyzed 13,500 free Android apps for MITM vulnerabilities. They found that 8% of the analyzed apps contain potentially vulnerable TLS modules. They also performed manual inspection of 100 apps, and successfully executed MITM attacks on 41, capturing credentials for widely used commercial and social websites, e.g., Google, Facebook, Twitter, Paypal, and several banks. Their attacks relied on exploiting flaws in the certificate validation process; many apps ignored the chain of trust validation, accepting self-signed certificates, and mismatched common names.

Georgiev et al. [44] demonstrated that several widely used applications and development libraries, such as Amazon's EC2 Java library, Amazon and Paypal's SDK, osCommerce, and Java web services, among others, suffered from certificate validation vulnerabilities, leading to generic MITM attacks. These vulnerabilities were attributed to be caused by (primarily) the use of poorly designed APIs, such as JSSE and OpenSSL.

Brubaker et al. [37] designed an automated approach for testing the certificate validation modules of several well-known TLS implementations. They first scanned the Internet for servers with port 443 open using ZMap [31], and collected all the available certificates. Then, they permuted the certificate parameters and possible X509 values, compiling a list of 8 million *Frankencerts*. Using Frankencerts and differential testing, Brubaker et al. found over 200 discrepancies in these commonly used TLS implementations (e.g., OpenSSL, GnuTLS and NSS). He et al. [45] designed an automated static analysis tool for analyzing TLS libraries and applications. They then evaluated Ubuntu 12.04 TLS packages, and found 27 zero-day TLS vulnerabilities, related to faulty certificate/hostname validation.

Sivakorn et al. [52] proposed a black-box hostname verification testing framework for TLS libraries and applications. They evaluated the hostnames accepted by seven TLS libraries and applications, and found eight violations, including: invalid hostname characters, incorrect null characters parsing, and incorrect wildcard parsing.

Chau et al. [38] made use of a symbolic execution approach to test the certificate validation process of nine TLS libraries, compared to RFC 5280 [46]. They found 48 instances of noncompliance; libraries ignored several X509 certificate parameters, such as the pathLenConstraint, keyUsage, extKeyUsage, and 'notBefore' validity dates.

Comparison. The most closely related work is by Durumeric et al. [42] (other studies mostly involved analyzing TLS libraries and client-end proxies). While their work focuses primarily on fingerprinting TLS interception, in addition to a brief security measurement for several HTTPS proxies, we develop an extensive framework dedicated for analyzing the TLS interception on network appliances. They checked/rated the highest TLS version supported by a target proxy, while we examine all the supported versions by the proxy, in addition to their respective mapping/mirroring to the client side. Durumeric et al.'s certificate validation tests include: expired, self-signed, invalidly signed certificates, and certificates signed by CAs with known private keys; we include more tests for this important aspect (a total of 32 distinct tests). We also include several new tests such as: checking the content of the CA trusted store and the certificate parameter mapping, locating the private signing keys of the proxies and examining their security (including checking pre-generated root certificates); these tests are mostly added/extended from [38, 39]. In terms of results, for the five products overlapping with Durumeric et al. [42], we observed a few differences; see Section 7.

4 PROPOSED FRAMEWORK

In this section, we present the setup/architecture of the proposed framework, and its major components and tests.

4.1 Test Setup/Architecture

Our framework consists of three virtual machines: a client, a web server, and the TLS intercepting network appliance; see Figure 1. The client machine (Windows 7 SP1) is located behind the appliance; we update the client with all available Windows updates, and install up-to-date Mozilla Firefox,

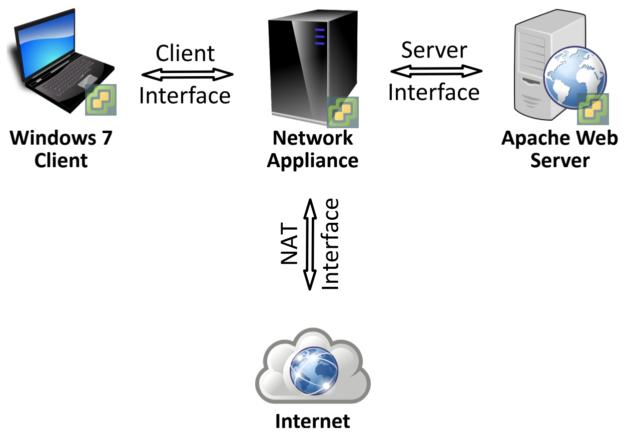

Fig. 1. Framework components and the overall test architecture (we use virtual machines for the client PC, network appliance and web server)

Google Chrome, and Internet Explorer 11 on it. We insert the TLS proxy's root certificate into the client's trusted CA stores (both Windows and Mozilla stores). We use a browser to initiate HTTPS requests to our local Apache web server, and the online TLS security testing suites (for certain tests). These requests are intercepted by the tested TLS proxy.

The second machine hosts a web server (Apache/Ubuntu 16.04), and accepts HTTP and HTTPS requests (on ports 80 and 443, respectively); all port 80 requests are redirected to port 443. It is initially configured to accept all TLS/SSL protocol versions, and all available cipher suites. The server name is configured to be *apache.host*, as the crafted certificates must hold a domain name (not an IP address). We generate the faulty certificates using OpenSSL, which are served from the web server. It also hosts the patched howsmyssl.com code [12].

The pre-installed OpenSSL version on the Ubuntu 16.04 distribution is not compiled with SSLv3 support. Thus, in order to test the acceptance and mapping of SSLv3 only, we rely on an identically configured older version of Ubuntu (14.04), with an older OpenSSL version that supports SSLv3.

The third machine hosts the appliance that we want to test. The appliances are typically available as a trial version on a vendor's website, with a pre-configured OS, either as an ISO image or an Open Virtualization Format file. The appliances are configured to intercept TLS traffic either as a transparent or explicit proxy, depending on the available modules. If both are available, transparent proxies are prioritized, as they do not require any client-side network configuration. We disable services such as firewall and URL filtering, if bundled in the appliances, to avoid any potential interferences in our TLS analysis. The root CA certificates corresponding to our faulty test certificates are injected into the trusted stores of the appliances. We include the following TCP ports to the list of intercepted ports, as they are used by the Qualys client test and badssl.com: 1010, 1011, 10200, 10300, 10301, 10302, 10303, 10444, and 10445 (determined by analyzing traffic captures on Wireshark of TLS connections to Qualys/badssl websites). Some appliances offer an interface to add custom ports to be intercepted by the TLS proxy, while others require manual configuration in their configuration files. We performed several rounds of updates and patches for Microsoft, on a Windows Server 2008 R2 operating system, as recommended by Microsoft's documentation [16]. These include the service pack 1 (SP1), the service pack 1 update, the service pack 2, and five rollup updates (1 to 5) [15].

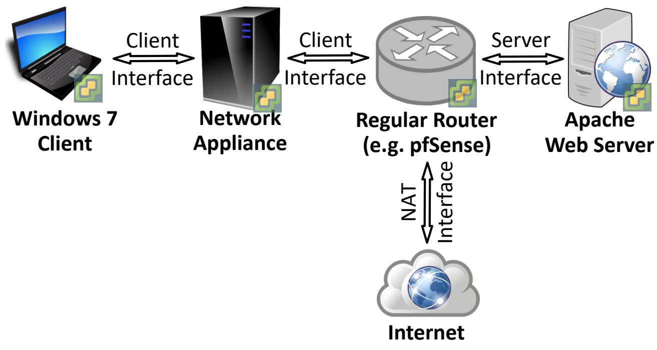

Fig. 2. Framework components and test architecture with a router (virtual machines)

We set up a local DNS entry for *apache.host* on the client, web servers and network appliances machines. Operating systems match local DNS entries, found typically in the *hosts* file, before remote DNS entries, resulting in the correct mapping of our test server's domain name to its corresponding IP address.

We require three different interfaces on each virtual network appliance. The *Client Interface* is used to connect to the Windows 7 client. The traffic incoming from this interface is intercepted by the TLS proxy. Transparent proxies only require the appliance to be the default gateway for the client, while explicit proxies require the client to configure the proxy settings with the appliance's socket details. The *Server Interface* is used to connect to the Apache web server. The *WAN Interface* connects to the Internet, through Network Address Translation (NAT). However, some appliances support one or two interfaces. In such cases, we add a fourth virtual machine, that acts solely as a router with multiple interfaces. We use pfSense as the router (without TLS interception), relying on it for NATting and routing traffic of the three interfaces as required; see Figure 2. The client and the network appliance are connected to the *Client Interface*, the web server is connected to the *Server Interface*, and the Internet connectivity is provided through NAT on a third interface via pfSense. A local DNS entry for *apache.host* is also added to this router.

4.2 Trusted CA Store

We first locate the trusted store of the TLS proxy, to inject our root certificates in it, required for most of our tests.

Injecting custom certificates into a trusted store could be trivial, if the appliance directly allows adding custom root CAs (e.g., via its user interface). If no such interface is offered, we attempt to get a command line (*shell*) access through a terminal, or, the SSH service if available, by enabling the SSH server first through the settings panels (we transfer files using the SCP/SFTP protocols). If SSH is unavailable, we mount the virtual disk image of the appliance on a separate Linux machine. When mounting, we perform several attempts to find the correct filesystem type and subtype used by the appliance (undocumented). After a successful mount, we search the entire filesystem for digital certificates in known formats, such as ".crt", ".pem", ".cer", ".der", and ".key". We thus locate several directories with candidate certificates, and subsequently delete the content of each file, while trying to access regular websites from the client. When an "untrusted issuer" warning appears at the client, we then learn the exact location/directory of the trusted store (and can eliminate duplicate/unnecessary certificates found in multiple directories).

We then inject our custom crafted root certificates into the trusted CA stores. We also parse the certificates available in the trusted stores to identify any expired certificates, or certificates with short key lengths (e.g., RSA-512 and RSA-1024). We also check for the presence of root CA certificates from issuers that are no longer trusted by major browser/OS vendors. Our list of misbehaving CAs includes: China Internet Network Information Center (CNNIC [6]), TÜRKTRUST [24], ANSSI [20], WoSign [7], Smartcom [7], and Diginotar [5].

4.3 TLS Version Mapping

To test the SSL/TLS version acceptance and TLS parameter mapping/mirroring, we alter the Apache web server's configuration. We use a valid certificate whose root CA certificate is imported into the trusted stores of the client (to avoid warnings and errors). We then subsequently force one TLS version after another at the web server, and visit the web server from the client, while documenting the versions observed in the browser's HTTPS connection information. Using this methodology, we are able to analyze the behavior of a proxy regarding each SSL/TLS version: if a given version is blocked, allowed, or altered in the client-to-proxy HTTPS connection.

4.4 Certificate Parameters Mapping

We check if the proxy-to-server certificate parameters are mapped or mirrored to the client-to-proxy certificate parameters. The parameters studied are signature hashing algorithms, certificate key lengths, and the EV/DV status.

For testing signature hashing algorithms, we craft multiple valid certificates with different *secure* hash algorithms, such as SHA-256, SHA-384 and SHA-512. We import their root CA certificates into the trusted stores of the client to avoid warnings and errors. We subsequently load each certificate and its private key into the web server, and visit the web page from the browser. We track the signature algorithms used in the certificates generated by the TLS proxy for each connection, and learn if the proxy mirrors the signature hashing algorithms, or use a single hard-coded one.

For testing certificate key lengths, we craft multiple certificates with multiple acceptable key sizes: RSA-2048, RSA-3072 and RSA-4096. We import their correspondent root CA certificates into the trusted stores of the client. We subsequently load each certificate and its private key into the web server, and visit the web page from the browser. We check the key length used for the client-to-proxy server certificate generated by the proxy for each connection, and learn if the proxy mirrors the key-length, or uses a hard-coded length.

We rely on Twitter's website to study the network appliance's behavior regarding Extended Validation (EV) certificates. We visit twitter.com on the client machine, and check the client-to-proxy certificate displayed by the browser. TLS proxies can identify the presence of EV certificates, e.g., to avoid downgrading them to domain-validated (DV), by parsing the content and locating the CA/browser forum's EV OID: 2.23.140.1.1 [9].

4.5 Cipher Suites

Cipher suites offered by the TLS proxy in the proxy-to-server TLS connection can be examined in multiple ways. We initially rely on publicly hosted TLS testing suites, howsmyssl.com and the Qualys client test [23]. Since the connection is proxied, the displayed results found on the client's browser are the results of the proxy-to-server connection, and not the client-to-proxy connection. If the mentioned web pages are not filtered, for reasons such as the use unfiltered or non-standard ports, we use Wireshark to capture the TLS packets and inspect the Client Hello message initiated by the proxy to locate the list of ciphers offered.

We then compare the list of ciphers offered by the proxy to that list of our browsers, to deduce if the TLS proxy performs a cipher suite mirroring or uses a hard-coded list. We also parse the proxy's cipher-suite for weak and insecure ciphers that could lead to insecure and vulnerable TLS connections.

4.6 Known TLS Attacks

We test TLS proxies for vulnerabilities against well-known TLS attacks, including: BEAST, CRIME, FREAK, Logjam, and Insecure Renegotiation (see Appendix C). We rely on the Qualys SSL Client Test [23] to confirm if the TLS proxy is patched against FREAK, Logjam, and Insecure Renegotiation, and check if TLS compression is enabled for possible CRIME attacks. We visit the web page from the client browser, which displays the results for the proxy-to-server TLS connection. For the BEAST attack, we rely on howsmyssl.com [12] (with the modifications from [39]) to test the proxies that support TLS 1.2 and 1.1. For a system to be vulnerable to BEAST [40], it must support TLS 1.0, and use the CBC mode. However, after the BEAST attack was uncovered, a patch was released for CBC (implementing the $1/(n-1)$ split patch [1]), but was identically named as CBC, making the distinction between the patched/unpatched CBC difficult.

4.7 Crafting Faulty Certificates

We use OpenSSL to craft our invalid test certificates, specifying *apache.host* as the Common Name (CN), except for the wrong CN test. We then deploy each certificate on our Apache web server, and request the HTTPS web page from the proxied client, and thus learn how the TLS proxy behaves when exposed to faulty certificates; if a connection is allowed, we consider the proxy is at fault. If the proxy replaces the faulty certificate with a valid one (generated by itself), leaving no way even for a prudent client (e.g., an up-to-date browser) to detect the faulty remote certificate, we consider this as a serious vulnerability. If the proxy passes the unmodified certificate and relies on client applications to react appropriately (e.g., showing warning/error messages, or terminating the connection), we still consider the proxy to be at fault for two reasons: (a) we do not see any justification for allowing plain, invalid certificates by any TLS agent, and (b) not all TLS client applications are as up-to-date as modern browsers, and thus may fail to detect the faulty certificates.

When the certificate's chain of trust contain intermediate certificate(s), we place the leaf certificate and intermediate certificate(s) at the web server, by appending the intermediate certificate(s) public keys after the server leaf certificate, in *SSLCertificateFile*. Note that we inject the issuing CA certificates of the crafted certificates into the TLS proxy's trusted store for all tests, except for the unknown issuer test and the fake GeoTrust test.

We compile the list of faulty certificates using several sources (including [38, 39, 46]; see Appendix A for details). Before using the faulty certificates, we assess them against Firefox v53.0 (latest at the time of testing), and confirm that Firefox terminates all connections with these certificates.

As part of the analysis of the certificate validation mechanisms, we ensure that the TLS proxies do not cache TLS certificates, by checking the 'Organization Name' field of the subject parameter in the server certificates. Each leaf certificate of the crafted chains contains a unique 'Organization Name' value, allowing us to identify exactly which TLS certificate is being proxied. We additionally check if the TLS inspection feature is enabled by default after the activation of the appliances, or if it requires a manual activation.

4.8 Private Key Protection, Self-issued, and Pre-Generated Certificates

We attempt to locate a TLS proxy's private key (corresponding to its root certificate), and learn if it is protected adequately, e.g., inaccessible to non-root processes, encrypted under an admin password. Subsequently, we use the located private keys to sign leaf certificates, and check if the TLS proxy accepts its own certificates as the issuing authority for externally delivered content.

To locate the private keys on the non-Windows systems, access to the network appliances' disks content and their filesystems is required. If we get access to an appliance's filesystem (following Section 4.2), we search for files with the following known private key file extensions: ".pem", ".key",

Table 2. Results for TLS parameter mapping/mirroring and vulnerabilities to known attacks. For TLS version mapping, we display the TLS versions seen by the client when the web server uses TLS 1.2, 1.1, 1.0 and SSL 3.0 ('–' means unsupported. '†' means supported but terminate with a handshake failure; see Section 5.1). Under "Key Length Mapping": '*' means the appliance mirrors RSA-512 and RSA-1024 key sizes, but use a static key size RSA-2048 for any higher key sizes (see Section 5.1). Under "BEAST": ✗ means vulnerable; ✗* means potentially vulnerable (unknown if CBC is patched with $1/(n-1)$ split); blank means patched. All the appliances are patched against FREAK, Logjam, CRIME, and Insecure Renegotiation.

	TLS Version Mapping				Cipher Suites		Certificate Parameter Mapping			
	Server TLS 1.2	Server TLS 1.1	Server TLS 1.0	Server SSL 3.0	Cipher Suites Mirroring	Weak/Broken Ciphers, Hash Algorithms	Key Length Mapping	Signature Algorithm Mapping	EV Certi-ficates	BEAST
pfSense	1.2	–	–	–	✗		2048	SHA256	DV	
OpenSense	1.2	1.2	1.2	–	✓		2048	SHA256	DV	✗*
Endian	1.2	1.2	1.2	1.2	✗	3DES, **RC4**, IDEA	2048	SHA256	DV	✗*
Untangle	1.2	1.2	1.2	–	✗	3DES	2048	SHA256	DV	✗*
WebTitan	1.2	1.2	1.2	1.2	✗	3DES, **RC4**, IDEA	1024	SHA256	DV	✗*
UserGate	1.2	1.2	1.2	–	✗	3DES, **DES**	1024	SHA256	DV	✗*
Cisco	1.2	1.2	1.2	–	✗		2048*	SHA256	DV	✗
Sophos	1.2	1.2	1.2	–	✗		2048	SHA256	DV	✗*
TrendMicro	1.2	†	†	–	✗	3DES, **RC4**	1024	SHA256	DV	✗
McAfee	1.2	1.2	1.2	–	✗		2048	SHA256	DV	
Cacheguard	1.2	1.2	1.2	–	✗	3DES	2048	SHA256	DV	✗*
Comodo	1.2	1.2	1.2	1.2	✗	3DES, **RC4**, IDEA	2048	SHA256	DV	✗*
Microsoft	–	–	1.0	3.0	✗	3DES, **DES**, **RC4**, **MD5**	2048	Mirrored	DV	✗

".pfx", and ".p12", and then compare the modulus of located RSA private keys with the proxy's public key certificate to locate the correct corresponding key. Alternatively, we locate the 'squid.conf' file, the configuration file of the Squid [22] proxy, used by most appliances as the proxy API. Squid is an open source proxy, that performs TLS interception through its 'ssl_bump' option. The configuration file points to the full path of the private key, and thus, leads us to the location of the RSA key. If the filesystem is inaccessible, we parse the raw disks for keys, using the Linux command *strings* on the virtual hard disk file and search for private keys. We also use memory analysis tools, such as Volatility [29] and Heartleech [11], to extract the private keys in some cases; for more information, see Appendix B. If we acquire the private key using this methodology, we still get no information on the key's location within the appliance's file system, storage method (e.g., encrypted, obfuscated), and privileges required to access the key. For Windows-based appliances, we utilize Mimikatz [18] to extract the private key (cf. [39]). Key storage is usually handled on Windows using two APIs: Cryptography API (CAPI), or Cryptography API: Next Generation (CNG [30]). When executed with Administrator privileges, Mimikatz exports private keys that are stored using CAPI and CNG. We check the location of the private keys, the privilege required to access them and if any encryption or obfuscation is applied. If the located private key on disk is encrypted, we rely on a python script to launch a dictionary attack.

We also check if appliance vendors rely on pre-generated certificates for their proxies, which could be very damaging (e.g., trivial MITM attacks [39]). We install two instances of the same product, and compare the certificates along with their correspondent private keys (if located). If we find the same key, we conclude that the appliance uses a pre-generated certificate, instead of per-installation keys/certificates.

5 RESULTS

In this section, we detail the results of our analysis on TLS parameters, certificate validation, trusted certificate stores, and private key protection.

5.1 TLS Parameters

Table 2 shows an overview of our results.

TLS versions and mapping. Ten appliances support TLS versions 1.2, 1.1, and 1.0, among which three also support SSL 3.0. pfSense supports TLS 1.2 only (restricting access to some sites). Microsoft supports only TLS 1.0 and (more worryingly) SSLv3; as many web servers nowadays do not support these versions (specifically SSLv3), clients behind Microsoft will be unable to visit these websites (Over 25% of web servers do not support TLS 1.1 & TLS 1.2. [14]).

TrendMicro terminates the TLS connections if the highest TLS version supported by the client is not supported by the requested server, instead of using a lower TLS version that is supported by both the client and the server. For example, if the requesting client supports TLS versions 1.2, 1.1 and 1.0, and the requested server supports TLS 1.1 and 1.0 only, TrendMicro terminates the connection (with a handshake failure) instead of establishing it with the TLS version 1.1. This behavior is a more restrictive form of TLS version mirroring. Except Microsoft and TrendMicro, other appliances map all the proxy-to-server TLS versions to TLS 1.2 for the client-to-proxy connection, and thus mislead browsers/users through this artificial version upgrade.

Certificate parameters and mapping. No appliance, except Cisco, mirrors the RSA key sizes; instead, they use a hard-coded key length for all generated certificates (i.e., artificially upgrade/downgrade the external key-length to RSA-2048, and thus may mislead clients/users). When exposed to RSA-512 and RSA-1024 server certificates, Cisco mirrors those key lengths to client-to-proxy TLS connection. However, when exposed to RSA-2048, RSA-4096 and RSA-8196, Cisco maps those key lengths to a static RSA-2048 key size for the client-to-proxy TLS connection. Three appliances use the currently non-recommended RSA-1024 certificates [33]. Microsoft is the only appliance which mirrors the hash algorithm; the remaining appliances use SHA256, thus making external SHA1-based certificates (considered insecure) invisible to browsers.

All appliances intercept TLS connections with EV certificates, and thus, inevitably downgrade any EV certificate to DV (as the proxies cannot generate EV certificates).

Table 3. Results for certificate validation, part I. ✗ means a faulty certificate is accepted and converted to a valid certificate by the TLS proxy; ⇢ means a faulty certificate is accepted by the TLS proxy but caught by the client browser (Firefox); and blank means certificate blocked. Endian* does not have certificate validation enabled by default.

	Self Signed	Signature Mismatch	Fake Geo-Trust	Wrong CN	Unkn-own Issuer	Non-CA Interm-ediate	X509v1 Interm-ediate	Invalid pathLen-Constraint	Bad Name Constraint Intermediate	Unknown Critical X509v3 Extension	Malformed Extension Values
pfSense									✗		⇢
OpenSense									✗		⇢
Endian*									✗		⇢
Untangle				⇢					✗		✗
WebTitan	✗	✗	✗	⇢	✗	✗	✗	✗	✗	✗	⇢
UserGate	✗	✗	✗	⇢	✗	✗	✗	✗	✗	✗	⇢
Cisco				⇢					✗		
Sophos									✗		✗
TrendMicro									✗		✗
McAfee									✗	✗	✗
Cacheguard	✗								✗		⇢
Comodo	⇢	✗	✗	⇢	✗	✗	✗	✗	✗	✗	⇢
Microsoft									✗		✗

Cipher suites. Only OpenSense mirrors the client's cipher suites to the server side. Each of our test client's (Chrome/Firefox/IE) own list of cipher-suite is displayed on the Qualys test when the connection is proxied by OpenSense.

All other appliances use a hard-coded list of cipher suites instead; only four offer cipher suites that exclude any weak ciphers or hash algorithms. Eight of them offer 3DES, now considered weak due its relatively small block size [35]. Two appliances offer the insecure DES cipher [55]. Five appliances include the RC4 cipher, which has been shown to have biases [56], and is no longer supported by any modern browsers. Microsoft includes the deprecated MD5 hash algorithm [58]. Three appliances offer IDEA ciphers [36] with a 64-bit block length. When relying on the DHE ciphers, a reasonably secure modulus value should be used, e.g., 2048 or higher [32]. However, eleven appliances accept a 1024-bit modulus; UserGate and Comodo even accept a 512-bit modulus.

Table 4. Results for certificate validation, part II. For legend, see Table 3; N/A means not tested as the appliance disallows adding the corresponding faulty CA certificate to its trusted store.

	Revoked	Expired Leaf	Expired Interm-ediate	Expired Root	Not Yet Valid Leaf	Not Yet Valid Interm-ediate	Not Yet Valid Root	Leaf keyUsage w/out Key Enciph-erment	Root keyUsage w/out KeyCert-Sign	Leaf extKey-Usage w/ clientAuth	Root extKey-Usage w/ Code Signing
pfSense	✗										✗
OpenSense	✗										✗
Endian*	✗										✗
Untangle	✗			✗			✗		✗		✗
WebTitan	✗	⇢	✗	✗	⇢	✗	✗	⇢	✗	⇢	✗
UserGate	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
Cisco		⇢	✗	N/A	⇢	✗	N/A		N/A		N/A
Sophos	✗										✗
TrendMicro	✗										✗
McAfee								✗		✗	✗
Cacheguard	✗										✗
Comodo	✗	⇢	✗	✗	⇢	✗	✗	⇢	✗	⇢	✗
Microsoft								✗			✗

5.2 Certificate Validation Results

In this section, we discuss the vulnerabilities found in the certificate validation mechanism of the tested TLS proxies; for summary, see Tables 3, 4 and 5.

WebTitan, UserGate and Comodo do not perform *any* certificate validation; their TLS proxies allowed *all* our faulty TLS certificates. UserGate enables TLS inspection by default after a fresh installation. Endian does not perform certificate validation by default (a checkbox for accepting all certificates is checked by default). We uncheck the checkbox to test the certificate validation mechanism in Endian, and discuss the results based on the forced certificate validation. Comodo also includes in its configuration interface a checkbox for accepting all certificates, checked by default. Even after unchecking it, the appliance still does not perform any certificate validation.

Both WebTitan and UserGate block access to the web servers offering RSA-512 certificates, possibly triggered by the TLS libraries utilized by the proxies, and not by the TLS interception certificate validation code (as apparent from the error messages we observed). Although Comodo accepts self-signed certificates, Firefox caught the faulty certificate. This is the result of Comodo mirroring the X.509 version 3 extension 'basic constrains: CA' value of the server self-signed certificate to the client-side TLS connection. Note that Firefox blocks a TLS connection when the delivered leaf certificate has the CA flag set to true, while Chrome accepts it. We omit WebTitan, UserGate and Comodo from the remaining discussion here, as they do not perform any certificate validation.

UserGate and TrendMicro cache TLS certificates and ignore changes in the server-side certificates (as opposed to modern browsers). Therefore, we regenerate a key pair for their TLS proxies for each of our certificate validation test, to ensure accuracy (i.e., not the results of cached TLS certificates).

We mark a faulty certificate as *passed* when the TLS proxy accepts the faulty certificate but leaves some chances for a diligent client to catch the anomaly. This behavior results from the way TLS proxies mirror X.509 extensions and their values to the client-to-proxy connection. The parameters mirrored are typically the common name, the keyUsage and extKeyUsage extensions, and the not before and expiry dates. In addition, Comodo mirrors the basic constraints CA flag, Cisco mirrors the RSA key size when it is 1024 bits and lower, and Microsoft mirrors the signature hashing algorithm. For simplicity, we report our results using the Firefox browser, but some results may change based on the client's validation process. For example, the Chrome browser allowed the leaf certificate with the basic constraints CA flag set to true in the Comodo self-signed test, while Firefox blocked access in this case.

Cacheguard accepts self-signed certificates (explicitly allowed in its default configuration). Untangle and Cisco forward the wrong CN certificates to our Firefox browser, which caught it and blocked access. McAfee is the sole appliance to accept a leaf certificate with an unknown x509 version 3 extension, marked as critical. Regarding malformed extension values, only Cisco blocks the anomalous certificate; pfSense, Cacheguard, OpenSense, and Endian pass it to the browser. Only Microsoft, Cisco, and McAfee check the revocation status of the offered certificates. When exposed to expired or not yet valid leaf certificates, Cisco forwards the certificates to the browser, as its default settings are configured to only monitor expired leaf certificates, and not to drop the connections.

Untangle fails to detect expired or not yet valid root CA certificates; Cisco disallows adding them to its trusted store in the first place. Cisco fails to detect expired and not yet valid intermediate certificates. Microsoft and McAfee allow leaf certificates whose keyUsage do not include keyEnciphernment. Untangle fails to detect root CA certificates that do not have keyCertSign among the keyUsage values, and Cisco disallows adding them to its trusted store. Similarly, Cisco disallows adding root CA certificates whose extKeyUsage parameter is codeSigning and RSA-512 root CA certificates to its store. McAfee accepts leaf certificates whose extKeyUsage x509 version 3 parameter is set to clientAuth.

Table 5. Results for certificate validation, part III. For legend, see Table 3.

	Root Key Length (Good Leaf)		Leaf Key Length (Good Root)				Signature Hashing Algorithm			DHE Modulus Length		Own Root
	512	1024	512	768	1016	1024	MD4	MD5	SHA1	512	1024	
pfSense	✗	✗		✗	✗	✗		✗	✗		✗	
OpenSense	✗	✗	✗	✗	✗	✗		✗	✗		✗	
Endian*	✗	✗	✗	✗	✗	✗		✗	✗		✗	
Untangle	✗	✗		✗	✗	✗			✗		✗	
WebTitan	✗	✗		✗	✗	✗	✗	✗	✗		✗	✗
UserGate	✗	✗		✗	✗	✗	✗	✗	✗	✗	✗	✗
Cisco	N/A	✗	➙	✗	✗	✗	✗	✗	✗		✗	
Sophos	✗	✗	✗	✗	✗	✗		✗	✗			
TrendMicro	✗	✗		✗	✗	✗		✗	✗		✗	
McAfee	✗	✗										
Cacheguard	✗	✗	✗	✗	✗	✗		✗	✗		✗	
Comodo	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
Microsoft		✗				✗	➙	➙	➙		✗	✗

Sophos, Cacheguard, OpenSense, and Endian accept RSA-512 leaf certificates (easily factorable [54]), and then issue certificates with RSA-2048, leaving no options for browsers to catch such certificates. Cisco also allows RSA-512 certificates, but Firefox detects them, as Cisco's proxy mirrors the

RSA key sizes of RSA-512 and RSA-1024 server certificates to the client-to-proxy TLS connection (RSA-2048 and higher key sizes are mapped to RSA-2048).

Microsoft mirrors signature hashing algorithms, and thus passes weak and deprecated hash algorithms (if any) to the client. Cisco accepts certificates signed using the deprecated MD4 algorithm. Microsoft, WebTitan, UserGate, and Comodo fail to detect external leaf certificates signed by their own root keys. Note that, when a TLS connection is terminated, Untangle and Microsoft use a TLS handshake failure; pfSense, Sophos, TrendMicro, McAfee, Cacheguard, OpenSense, and Endian redirect the connection to an error page; and Cisco uses an untrusted CA certificate, relying on the browser to block the connection. However, error pages as displayed by Sophos and TrendMicro, allow end-users to reestablish the connection (Sophos through 'Add exception for this URL', and TrendMicro through 'Continue at own your risk'). This behavior is a deviation from current practice (in browsers), as the users may be unaware of the actual risks and consequences if they bypass these warnings.

Table 6. Results for trusted stores, private keys and initial setup. 'N/A': not available (failed to locate the private key on disk); 'Not Applicable': the appliance does not rely on any store (no certificate validation); 'World': readable by any user account on the appliance.

	Trusted CA Store		Private Key			Initial Behavior	
	Location	Type	Location	State	Read Permission	Inspection By Default	Pre-Generated Key Pair
pfSense	/usr/local/share /certs/ca-root-nss.crt	Mozilla NSS	/usr/local/etc /squid/serverkey.pem	Plaintext	World	Off	No
OpenSense	/usr/local/openssl/cert.pem	Mozilla NSS	/var/squid/ssl/ca.pem	Plaintext	Root	Off	No
Endian	/etc/ssl/certs/ca-certificates.crt	'update-ca-certificates' Command	/var/efw/proxy/https_cert	Plaintext	World	Off	No
Untangle	/usr/share/untangle/lib/ ssl-inspector/trusted-ca-list.jks	Java Key Store	/usr/share/untangle /settings/untangle-certificates/untangle.key	Plaintext	Root	Off	No
WebTitan	Not Applicable	None	/usr/blocker/ssl /ssl_cert/webtitan.pem	Plaintext	World	Off	No
UserGate	Not Applicable	None	/opt/entensys/webfilter /etc/private.pem	Plaintext	World	On	No
Cisco	Network → Certificate Management → Cisco Trusted Root Certificate List	GUI	N/A	N/A	N/A	Off	No
Sophos	Web Protection → Filtering Options → HTTPS CAs → Global Verification CAs	GUI	N/A	N/A	Admin (for GUI download)	Off	No
TrendMicro	HTTP → Configuration → Digital Certificates → Active Certificates	GUI	/var/iwss/https/certstore /https_ca/default_key.cer	Passphrase Encryption	World	Off	Yes
McAfee	Policy → Lists → Subscribed Lists → Certificate Authorities → Known CAs	GUI	N/A	N/A	Admin (for GUI download)	Off	Yes
Cacheguard	/usr/local/proxy/var /ca-ssl/ca-bundle.crt	Mozilla NSS	/usr/local/proxy/var /ca-ssl/self-ca.key	Plaintext	World	Off	Yes
Comodo	Not Applicable	None	/var/cni/credentials/ca.key	Plaintext	World	Off	No
Microsoft	mmc.exe → Windows Trusted Store → Local Computer	Microsoft Store	CERT_SYSTEM_STORE _LOCAL_MACHINE_MY	Exportable Key	Admin	Off	No

5.3 Trusted CA Stores

In this section, we analyze the results for trusted CA stores, their accessibility, source, and content; see Table 6. Note that, as WebTitan, UserGate, and Comodo perform no certificate validations, their trusted stores are of no use.

Accessing the trusted stores. Untangle's file system can be accessed through SSH. We found that Untangle relies on two CA trusted stores, saved in Java Keystore files on the filesystem. The first store, 'trusted-ca-list.jks', holds the CA authorities trusted by default, while the second store, 'trustStore.jks', holds the custom CA certificates, added by the machine administrator through Untangle's UI. pfSense also allows SSH, and we found that its CA trusted store on the FreeBSD filesystem under 'ca-root-nss.crt'. pfSense does not offer any UI to add custom CA certificates. We append our crafted certificates to the original store, in a format that includes the public key, in addition to the text meta-data (OpenSSL's '-text' option). Microsoft relies on the Windows Server's standard trusted store. To view the content of the trusted store and to inject our crafted CA certificates, we rely on the Microsoft Management Console, in the Trusted Root Certification Authorities section of the Local Computer.

Cisco's trusted CA store can be accessed through the appliance's web interface, under the Cisco Trusted Root Certificate List. It also includes another interface, the Cisco Blocked Certificates, for untrusted issuer certificates. To add custom CA certificates, the appliance includes a third interface, the Custom Trusted Root Certificates. However, Cisco does not allow the injection of most of our invalid root certificates, and responds with an error when tried. Sophos allows accessing the trusted CA store through its web interface, under Global Verification CAs. The interface allows adding custom root certificates, in addition to disabling CA certificates that are included by default. TrendMicro's trusted store can be accessed through the web interface's Active Certificates section. It is possible to add custom CA certificates and deactivate existing default ones. McAfee gives access to the root certificates supplied by default in the Known CAs section, and allows adding custom root certificates in the My CAs section.

Cacheguard's web interface does not include a section for root CA certificates. In addition, Cacheguard does not give access to its filesystem through a terminal, and does not support SSH. We thus mount the appliance's virtual hard disk to a Linux machine, and locate the trusted store in a 'ca-bundle.crt' file. We subsequently append our custom CA certificates to the bundle. Similarly, OpenSense and Endian do not include a section for root certificates. However, they give access to the filesystem through an OS shell terminal. We locate the trusted store of OpenSense in a 'cert.pem' bundle file, and Endian's in a 'ca-certificates.crt' bundle file. We include our custom CA certificates to these files.

Source and content. As documented on Untangle's SSL Insepctor wiki page [26], the list of trusted certificates is generated from the Debian/Linux ca-certificates package, in addition to Mozilla's root certificates. However, Untangle includes Microsoft's own Root Agency certificate, indicating the additional inclusion of Windows trusted certificates. The Root Agency certificate is RSA-512 that can be trivially compromised (see Section 6). Untangle also includes 21 RSA-1024 root certificates, 30 expired certificates, and 16 certificates from issuers that are no longer trusted by major browser/OS vendors (three from CNNIC CA, six DigiNotar, three TÜRKTRUST, and four WoSign certificates).

pfSense's trusted CA store relies on Mozilla's NSS certificates bundle, extracted from the nss-3.30.2 version (Apr. 2017), with 20 untrusted certificates omitted from the bundle, as specified in the header of the trusted store. It does not include any RSA-512 or RSA-1024 certificates, and no expired certificates. However, pfSense includes two CNNIC CA certificates, and four WoSign CA certificates.

Similar to the other Windows OSes, the Windows Server 2008 R2 also does not display the full list of trusted root certificates in its management console, and instead, only displays the root certificates of web servers already visited. We thus rely on the Microsoft Trusted Root Certificate Program [17] to collect the list of certificates trusted to the date of the testing. We found that the list includes two CNNIC CA certificates, two TÜRKTRUST CA certificates, two ANSSI CA certificates, and four WoSign CA certificates. Nonetheless, the acquired list does not include the RSA key sizes of the certificates, their expiry dates, or their revocation states.

As for Cisco, we found four problematic root CA certificates from TÜRKTRUST included into the trusted store. However, the RSA key sizes are not displayed within the UI, so we could not check for RSA-512 and RSA-1024 CA certificates.

Sophos includes two CNNIC, four WoSign, and three TÜRKTRUST CA certificates; TrendMicro has a CNNIC, two TÜRKTRUST, and 30 expired certificates; and McAfee includes a CNNIC certificate. The RSA key sizes (for all three) and expiry dates (for Sophos and McAfee) of CA certificates are not displayed within their UI, and thus we could not check for these issues.

Cacheguard's trusted store is extracted from Mozilla NSS's root certificates file 'certdata.txt' [19] and converted using Curl's 'mk-ca-bundle.pl' version 1.27 script [4], as specified in the 'ca-bundle.crt' trusted store file. We parse the trusted store using OpenSSL's '-text' option to extract the certificate metadata. The trusted store contains two TÜRKTRUST, four WoSign, three expired certificates; however, it is free of RSA-512 or RSA-1024 certificates. OpenSense's store also relies on Mozilla, extracted from the nss-3.35 (Jan. 2018) version, with two untrusted certificates omitted from the bundle, as specified in the header of the NSS trusted store. It does not include any RSA-512 or RSA-1024 certificates, and no expired certificates. However, it includes a TÜRKTRUST CA certificate.

Endian's trusted CA store bundle is the output of the 'update-ca-certificates' Debian Linux command [27]. The trusted store contains two CNNIC, three TÜRKTRUST, four WoSign, 10 expired, and 11 RSA-1024 CA certificates.

5.4 Private Key Protection

In this section, we discuss the results regarding the TLS proxies' private keys, in terms of storage location, state, and the privilege required to access them; see Table 6.

We could not access the filesystem of Cisco's AsyncOS to locate its private key on disk. Instead, we extract the key from memory using Heartleech [11] (see Appendix B). Sophos and McAfee provide access to their filesystems through a bash terminal. However, we could not locate their private keys on disk. Sophos stores the key in a database, as it can be recovered by invoking the following command 'cc get_object REF_CaSigProxyCa' via Sophos' terminal. McAfee possibly has its private key hard-coded, as its key pair is pre-generated, as discussed later in this section. Thus, we could not locate the private key on disk. We get a copy of their respective private keys by downloading them from the appliances' web interfaces. As there is no information on the private key on disk, and the located key was used only for testing external content signed by own key, we do not discuss these appliances in the rest of the section.

We rely on the methodologies from Section 4.2 to access the filesystems on non-Windows appliances. pfSense and Untangle provide SSH access. For WebTitan and Cacheguard, we mount their respective virtual disk disks on a separate machine. UserGate, TrendMicro, OpenSense, Comodo, and Endian provide access to their OS shell terminal by default. Untangle, pfSense, WebTitan, UserGate, Cacheguard, OpenSense, Comodo and Endian store their plaintext private keys within their filesystems (as 'untangle.key', 'serverkey.pem', 'webtitan.pem', 'private.pem', 'self-ca.key', 'ca.pem', 'ca.key', and 'https_cert' files, respectively). pfSense, WebTitan, UserGate, Cacheguard, Comodo, and Endian allow read access to all users accounts (write is restricted to root), while Untangle and OpenSense allow read/write only to root accounts.

Regarding TrendMicro, we get access to the filesystem using its OS terminal, and locate the root private key in a file named 'default_key', with read permission to all user accounts (write is restricted to root). However, the located key is encrypted using a passphrase. We brute-force the encrypted key using a python script and a dictionary of common English words, and successfully decrypt the key, with the passphrase 'trend'.

Microsoft's private key is stored using the Windows Software Key Storage Provider, utilizing Cryptography API: Next Generation (CNG). The key is exportable through the Microsoft Management Console, if opened with SYSTEM privileges. We rely on the Mimikatz tool to export the key, which requires a less privileged Administrator account.

We install multiple instances of each appliance to check if the root certificates are pre-generated. To our surprise, we found that TrendMicro, McAfee and Cacheguard use such certificates to intercept the TLS traffic. McAfee includes an X509v3 'Netscape Certificate Comment' extension, with the following warning: "This is the default McAfee root CA. It will be delivered with each web gateway installation. We recommend to generate and use your own CA." However, it does not provide any warning during installation/configuration. Although Cacheguard's documentation explicitly state: "the default system CA certificate is generated during the installation" [2], in reality, it uses a pre-generated certificate.

6 PRACTICAL ATTACKS EXPLOITING THE MAIN FINDINGS

In this section, we summarize how the main vulnerabilities reported here could be exploited by an attacker.

MITM attacks can be trivially launched to impersonate any web server against clients behind UserGate, WebTitan, Comodo and Endian, due to their lack of certificate validation (using default configuration). Attackers can simply use a self-signed certificate for any desired domain, fooling even the most secure and up-to-date browsers behind these appliances. Since Usergate enables TLS interception by default, users located behind a freshly installed UserGate appliance are automatically vulnerable. Likewise, clients behind Cacheguard are vulnerable, as the appliance's TLS proxy accepts self-signed certificates. Clients behind Untangle are also similarly vulnerable, due to the RSA-512 'Root Agency' certificate in its trusted store. The RSA-512 private key corresponding to this certificate can be easily factored under four hours [54] as a one-time effort, and the factored key could be used attack all instances of Untangle.

An attacker can also launch MITM attacks to decrypt traffic or impersonate any web server against clients behind TrendMicro, McAfee and Cacheguard, as they rely on pre-generated root keys (identical on all installations). The attacker can retrieve private keys for these appliances from her own installations irrespective of privileges required to access the keys.

Nine appliances do not encrypt their private keys, seven of which are accessible to unprivileged processes running on the same appliance. If an attacker could execute such an unprivileged process,[1] she could extract private keys to sign DV certificates for any server, enabling her to impersonate any web server. These unprotected private keys could be especially problematic for appliances that accept external certificates signed by their own root keys; such appliances include: UserGate, WebTitan, Microsoft, and Comodo. Microsoft protects their keys with admin privilge, but UserGate, WebTitan and Comodo provide 'read' access to non-root users.

When combined with a Java applet to bypass the same origin policy, the BEAST vulnerability [40] may allow an attacker to recover authentication cookies from the clients behind Microsoft, Cisco and TrendMicro. Attackers could also recover cookies from clients behind WebTitan, Microsoft, TrendMicro, Comodo, and Endian due to their use of RC4 [56].

[1]For example, by exploiting a vulnerability such as CVE-2014-4306 for WebTitan, which allows remote attackers to read arbitrary files; see https://nvd.nist.gov/vuln/detail/CVE-2014-4306.

Attackers could break session confidentiality for clients behind Sophos, Cacheguard, OpenSense, Comodo and Endian, as they accept RSA-512 external leaf certificates (RSA-512 is easily factored). Note that, in 2016, 1% of TLS web servers were found to host an RSA-512 certificate [54]. In contrast, modern browsers will refuse to establish such connections.

All appliances except Untangle and McAfee accept certificates signed using MD5, with WebTitan, Microsoft, UserGate, Cisco and Comodo also accept MD4. Weak collision resistance of MD5/MD4 [58] can be exploited in a practical attack scenario, where the attacker forges a rogue intermediate CA certificate that appears to be signed by a valid trusted root CA; all leaf certificates signed by this rogue CA will similarly be trusted by the appliances. As a result of this one-time effort, the holder of this rogue intermediate CA can launch MITM attacks and impersonate web servers, targeting the users behind all the appliances that accept certificates signed using MD5 [53].

7 EVOLUTION OF PRODUCTS BETWEEN 2016–2018

In this section, we highlight the evolution of the overlapping appliances that were tested in three separate instances between 2016 to 2018: by Durumeric et al. [42] in 2016 (disclosed to vendors in Sept. 2016), our own tests in 2017 (disclosed in Dec. 2017), and the latest product releases tested in 2018 as part of this paper (disclosed in May 2018).

In 2016, Untangle included RC4 and weak ciphers in its cipher-suite; we found that version 13.0 (2017) still included weak ciphers, but no RC4. The Untangle 13.2 release, tested in 2018, has no differences in its TLS interception processes compared to release 13.0, and thus, shows the exact same results. pfSense, which was not tested in 2016 by Durumeric et al., accepts the TLS version 1.1 in its 2.3.4 release (2017), while pfSense 2.4.2-P1 (2018) no longer does. Moreover, pfSense 2.3.4 maps the certificate keys to RSA-4096, while the latest version maps them to RSA-2048. In 2016, WebTitan had a broken certificate validation process and offered RC4 and modern ciphers; we found that WebTitan version 5.15 (2017) did not perform *any* certificate validation, was vulnerable to the CRIME attack, and still offered RC4, in addition to weak ciphers. Moreover, the latest version of WebTitan (5.16) in 2018 accepts SSLv3 (did not in 2017), but is now patched against CRIME. Microsoft performed no certificate validation in 2016 and the highest supported SSL/TLS version was SSLv2.0; it now (2018) performs certificate validation, and supports SSL versions 2.0, 3.0 and TLS 1.0. The Microsoft and UserGate product releases are the same in 2018 compared to 2017. Cisco no longer offers RC4 and export-grade ciphers, which was reported in 2016. Furthermore, Cisco build 270's CBC ciphers (2017) are not recognized by the Qualys client test, while the latest build's CBC ciphers (2018) are, indicating that the appliance is vulnerable to the BEAST attack. The older build fails to block RSA certificates with malformed extension values, while the latest build does. The latest build fails to block expired and not yet valid intermediate root certificates, in addition to RSA-512 leaf certificates, while the older build (270) blocks them successfully. In 2016, Sophos offered RC4, but not in the 2018 release.

We contacted the six affected companies after our 2017 tests, and received replies from three companies; Untangle replied with just an automatic reply, Entensys confirmed that they have passed the matter to its research team. Netgate (pfSense), stated that they philosophically oppose TLS interception, but include it as it is a commonly requested feature. Netgate also states that the TLS interception is done using the external package 'Squid', which it does not control completely. They claimed that our tested version was five releases old at that time. We found the latest version to have the exact same results, with two minor exceptions. We are also contacting all vendors from our latest 2018 tests.

Overall, the disclosures appear to have limited impact on vendors. Many vendors completely ignored the security issues (Untangle, Microsoft, UserGate, and pfSense). More worryingly, some products even became worse over time (Cisco), and some patched product releases introduced new vulnerabilities compared to their older versions (WebTitan).

8 CONCLUSION

We presented a framework for analyzing TLS interception behaviors of network appliances to uncover potential vulnerabilities introduced by them. We tested thirteen network appliances, and found that all their TLS proxies are vulnerable against the tests under our framework—at varying levels. Each proxy lacks at least one of the best practices in terms of protocol and parameters mapping, patching against known attacks, certificate validation, CA trusted store maintenance, and private key protection. We found that the clients behind the thirteen appliances are vulnerable to full server impersonation under an active MITM attack, of which one enables TLS interception by default. We also found that three TLS proxies rely on pre-generated root keys, allowing trivial MITM attacks.

While TLS proxies are mainly deployed in enterprise environments to decrypt the traffic in order to scan for malware and network attacks, they introduce new intrusion opportunities and vulnerabilities for attackers. As TLS proxies act as the client for the proxy-to-web server connections, they should maintain (at least) the same level of security as modern browsers; similarly, as they act as a TLS server for the client-to-proxy connections, they should be securely configured like any up-to-date HTTPS server, by default.

Before enabling TLS interception, concerned administrators may use our framework to evaluate their network appliances, and weigh the potential vulnerabilities that may be introduced by a TLS proxy against its perceived benefits. Vendors may also rely on our methodology to assess the security of their products and resort to up-to-date best practices. Client-side TLS parameters should be mirrored to the server-side connection, or at least enhanced, by making sure that the list of ciphers does not include weak/deprecated ciphers and is kept up-to-date. Certificate parameters such as the RSA key length should also be mapped to at least 2048 bits to avoid factoring attacks. The TLS libraries used by the proxies must be kept up-to-date, in order to protect the clients from known TLS attacks, in addition to MITM threats from bad certificate validation. Open-source TLS libraries, such as OpenSSL, must also be properly configured to detect and block weak root, intermediate and leaf key lengths, in addition to content signed by the proxy's own root key pair. The CA trusted store must be kept up-to-date, which could be achieved by simply pulling Mozilla's latest NSS certificates bundle on a regular basis, in addition to enabling certificate revocation checks. Root private keys must be encrypted, obfuscated and inaccessible to non-root processes. Moreover, vendors must consider applying these fixes in their default TLS inspection behavior, as some administrators might lack the technical knowledge to securely harden the appliances. Beyond fixing the security issues as we identified, vendors should also consider incorporating guidelines for secure TLS proxying as outlined in prior work (e.g., [39]).

ACKNOWLEDGEMENTS

We thank the ACM DTRAP and ASIA CCS 2018 anonymous reviewers for their helpful comments and suggestions that helped improve the paper's presentation, and Xavier de Carné de Carnavalet for his help during the initial development of our framework. This research is supported by NSERC.

REFERENCES

[1] BEAST attack 1/n-1 split patch. https://bugzilla.mozilla.org/show$_b$ug.cgi?id=665814#c59, Jul 2017.
[2] CacheGuard OS user's guide - SSL mediation. https://www.cacheguard.net/doc/guide/ssl$_m$ediation.html, Jan 2018.
[3] Cisco WSA AsyncOS documentation. https://www.cisco.com/c/en/us/products/security/email-security-appliance/asyncos$_i$ndex.html.

[4] Curl's mk-ca-bundle.pl - GitHub. https://github.com/curl/curl/blob/master/lib/mk-ca-bundle.pl, Jan 2018.
[5] DigiNotar CA breach. https://nakedsecurity.sophos.com/2011/09/05/operation-black-tulip-fox-its-report-on-the-diginotar-breach/.
[6] Distrusting new CNNIC certificates. https://blog.mozilla.org/security/2015/04/02/distrusting-new-cnnic-certificates/, Apr 2015.
[7] Distrusting new WoSign and StartCom certificates. https://blog.mozilla.org/security/2016/10/24/distrusting-new-wosign-and-startcom-certificates/, Oct 2016.
[8] Effects of HTTPS and SSL inspection on the client. https://vuls.cert.org/confluence/display/Wiki/Effects+of+HTTPS+and+SSL+inspection+on+the+client, Aug 2017.
[9] Extended validation OID. https://cabforum.org/object-registry/.
[10] GRC certificate validation revoked test. https://revoked.grc.com/.
[11] Heartleech - GitHub. https://github.com/robertdavidgraham/heartleech.
[12] Howsmyssl - GitHub. https://github.com/jmhodges/howsmyssl.
[13] Lenovo's superfish security. https://www.cnet.com/news/superfish-torments-lenovo-owners-with-more-than-adware/, Feb 2015.
[14] Mapping the current state of SSL/TLS - thesis. http://www.diva-portal.org/smash/get/diva2:1109739/FULLTEXT01.pdf, 2017.
[15] Microsoft TMG 2010 updates. https://blogs.technet.microsoft.com/keithab/2011/09/27/forefront-tmg-2010-service-pack-rollup-and-version-number-reference/.
[16] Microsoft TMG supported OS version. https://www.microsoft.com/en-ca/download/details.aspx?id=14238.
[17] Microsoft trusted root certificate program. https://gallery.technet.microsoft.com/Trusted-Root-Certificate-123665ca.
[18] Mimikatz - GitHub. https://github.com/gentilkiwi/mimikatz.
[19] Mozilla's 'certdata.txt' file. https://hg.mozilla.org/mozilla-central/raw-file/tip/security/nss/lib/ckfw/builtins/certdata.txt, Sep 2017.
[20] Revoking ANSSI CA. https://security.googleblog.com/2013/12/further-improving-digital-certificate.html, Dec 2013.
[21] The risks of SSL inspection. https://insights.sei.cmu.edu/cert/2015/03/the-risks-of-ssl-inspection.html, Mar 2015.
[22] SSL Bump configuration - Squid. http://www.squid-cache.org/Doc/config/ssl$_b$ump/.
[23] SSL client test. https://www.ssllabs.com/ssltest/viewMyClient.html.
[24] The TÜRKTRUST SSL certificate fiasco. https://nakedsecurity.sophos.com/2013/01/08/the-turktrust-ssl-certificate-fiasco-what-happened-and-what-happens-next/, Jan 2013.
[25] UFS - Linux Kernel archives. https://www.kernel.org/doc/Documentation/filesystems/ufs.txt.
[26] Untangle SSL inspector documentation. https://wiki.untangle.com/index.php/SSL$_I$nspector#Trust$_A$ll$_S$erver$_C$ertificates.
[27] update-ca-certificates - Debian System Manager's Manual. https://manpages.debian.org/jessie/ca-certificates/update-ca-certificates.8.en.html, Apr 2017.
[28] US-CERT alert on HTTPS interception. https://www.us-cert.gov/ncas/alerts/TA17-075A.
[29] Volatility. http://www.volatilityfoundation.org/26.
[30] Windows cryptography API (CNG). https://www.codeguru.com/cpp/w-p/vista/article.php/c13813/Windows-Cryptography-API-Next-Generation-CNG.htm.
[31] ZMap - GitHub. https://github.com/zmap/zmap.
[32] D. Adrian, K. Bhargavan, Z. Durumeric, P. Gaudry, M. Green, J. A. Halderman, N. Heninger, D. Springall, E. Thomé, L. Valenta, et al. Imperfect forward secrecy: How diffie-hellman fails in practice. In *ACM CCS*, Denver, CO, USA, 2015.
[33] E. Barker and A. Roginsky. NIST recommendations. *NIST Special Publication*, 800(131A):1–29, 2015.
[34] B. Beurdouche, K. Bhargavan, A. Delignat-Lavaud, C. Fournet, M. Kohlweiss, A. Pironti, P.-Y. Strub, and J. K. Zinzindohoue. A messy state of the union: Taming the composite state machines of tls. In *IEEE Symposium on Security and Privacy*, Fairmont, CA, USA, 2015.
[35] K. Bhargavan and G. Leurent. On the practical (in-) security of 64-bit block ciphers: Collision attacks on HTTP over TLS and OpenVPN. In *ACM CCS*, Vienna, Austria, 2016.
[36] E. Biham, O. Dunkelman, N. Keller, and A. Shamir. New attacks on IDEA with at least 6 rounds. *Journal of Cryptology*, 28(2):209–239, 2015.
[37] 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 *2014 IEEE Symposium on Security and Privacy*, pages 114–129, Fairmont, CA, USA, 2014.
[38] S. Y. Chau, O. Chowdhury, E. Hoque, H. Ge, A. Kate, C. Nita-Rotaru, and N. Li. SymCerts: Practical symbolic execution for exposing noncompliance in X.509 certificate validation implementations. In *IEEE Symposium on Security and Privacy*, Fairmont, CA, USA, 2017.

[39] X. de Carné de Carnavalet and M. Mannan. Killed by proxy: Analyzing client-end tls interception software. In *NDSS'16*, San Diego, CA, USA.
[40] T. Duong and J. Rizzo. Here come the ⊕ ninjas. *Technical Report*. http://www.hpcc.ecs.soton.ac.uk/~dan/talks/bullrun/Beast.pdf, May 2011.
[41] T. Duong and J. Rizzo. The CRIME attack. *Presentation at Ekoparty Security Conference*, 2012.
[42] Z. Durumeric, Z. Ma, D. Springall, R. Barnes, N. Sullivan, E. Bursztein, M. Bailey, J. A. Halderman, and V. Paxson. The security impact of HTTPS interception. In *NDSS'17*, San Diego, CA, USA.
[43] S. Fahl, M. Harbach, T. Muders, L. Baumgärtner, B. Freisleben, and M. Smith. Why eve and mallory love android: An analysis of Android SSL (in) security. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 50–61, Raleigh, NC, USA, 2012.
[44] 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 *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 38–49, Raleigh, NC, USA, 2012.
[45] B. He, V. Rastogi, Y. Cao, Y. Chen, V. Venkatakrishnan, R. Yang, and Z. Zhang. Vetting SSL usage in applications with SSLint. In *2015 IEEE Symposium on Security and Privacy*, pages 519–534, Fairmont, CA, USA, 2015.
[46] R. Housley, W. Ford, W. Polk, and D. Solo. RFC 5280: Internet x.509 public key infrastructure certificate and crl profile, May 2008.
[47] L. S. Huang, A. Rice, E. Ellingsen, and C. Jackson. Analyzing forged SSL certificates in the wild. In *2014 IEEE Symposium on Security and Privacy*, pages 83–97, Fairmont, CA, USA, 2014.
[48] J. Jarmoc. SSL/TLS interception proxies and transitive trust. *Black Hat Europe*, Mar 2012.
[49] M. O'Neill, S. Ruoti, K. Seamons, and D. Zappala. TLS proxies: Friend or foe? In *ACM IMC'16*, Santa Monica, CA, USA.
[50] E. Rescorla, M. Ray, S. Dispensa, and N. Oskov. RFC 5746: Transport layer security (TLS) renegotiation indication extension, Feb 2010.
[51] S. Ruoti, M. O'Neill, D. Zappala, and K. E. Seamons. User attitudes toward the inspection of encrypted traffic. In *USENIX SOUPS'16*.
[52] S. Sivakorn, G. Argyros, K. Pei, A. D. Keromytis, and S. Jana. HVLearn: Automated black-box analysis of hostname verification in SSL/TLS implementations. In *2017 IEEE Symposium on Security and Privacy*, pages 521–538, Fairmont, CA, USA, 2017.
[53] A. Sotirov, M. Stevens, J. Appelbaum, A. K. Lenstra, D. Molnar, D. A. Osvik, and B. de Weger. MD5 considered harmful today, creating a rogue CA certificate. In *Chaos Communication Congress*, 2008.
[54] L. Valenta, S. Cohney, A. Liao, J. Fried, S. Bodduluri, and N. Heninger. Factoring as a service. In *FC'16*, Barbados.
[55] P. Van De Zande. The day DES died. *SANS Institute*, Jul 2001.
[56] M. Vanhoef and F. Piessens. All your biases belong to us: Breaking RC4 in WPA-TKIP and TLS. In *USENIX Security Symposium*, pages 97–112, Washington D.C., USA, 2015.
[57] L. Waked, M. Mannan, and A. Youssef. To intercept or not to intercept: Analyzing TLS interception in network appliances. In *ACM ASIACCS'18*, Incheon, Korea.
[58] X. Wang and H. Yu. How to break MD5 and other hash functions. In *Eurocrypt'05*, Aarhus, Denmark.

A CRAFTING INVALID CERTIFICATE CHAINS

In this section, we detail the methodology used to create each certificate validation test.

Self-Signed. We generate a standalone certificate using OpenSSL with regular parameters.

Signature Mismatch. We first generate a regular CA certificate, and use it to sign a regular leaf certificate. We then modify the signature of the leaf public key certificate by flipping one of the last bits in the certificate. The certificate signature is positioned as the last item inside the certificate. We thus create a certificate with a mismatching signature, and test if the proxy validates the signature on the presented certificate.

Fake GeoTrust Global CA. We craft an issuing root certificate with the same certificate parameters as the GeoTrust Global CA authority. We mimic the Common Name field (CN = GeoTrust Global CA), the Organization field (O = GeoTrust Inc.), and the Country field (C = US). Before signing the leaf certificate, we remove the authority key identifier parameter from it. Without the authority key identifier, the certificate cannot be linked to its issuing certificate. By doing so, we test if the TLS proxy validates the chain of trust properly, or relies only on the certificate parameters such as the subject name alone.

Wrong Common Name (CN). We generate a regular root CA certificate, and use it to sign a regular leaf certificate that does not have *apache.host* as the value for Common Name field. If the TLS proxy accepts such a leaf certificate for the *apache.host* domain, then the proxy does not validate that the delivered certificate is for the exact domain requested, and thus, allows websites to impersonate other servers by using their valid certificate.

Unknown Issuer. The test relies on a normal issuing certificate and its normal leaf certificate. However, we do not import the issuing certificate to the trust store of the network appliance, and consequently, check if the TLS proxy is vulnerable to MITM attacks, when an attacker uses untrusted CA certificates as issuers for their certificate.

Non-CA Intermediate. We generate three certificates that serve respectively as the root CA, the intermediate certificate and the leaf certificate. However, we intentionally craft the intermediate certificate to have the basic constraint extension that identifies CA certificate as false. Using this methodology, we test if the TLS proxy ensures that the CA certificates have the ability to issue other certificates, using the CA flag. If the proxy does not detect such vulnerabilities, attackers could use any valid leaf certificate to sign other leaf certificates, and host them on their servers.

X509v1 Intermediate. The first version of x509 does not have the basic constraint extension, and thus, CA certificates cannot be differentiated from leaf certificates. As a result, x509v1 certificates should not be used for issuing certificates. We generate three certificates that serve respectively as the root CA, the intermediate certificate and the leaf certificate, while only having the intermediate certificate of type x509v1. If accepted, the proxy risks potential consequences that are similar to the Non-CA Intermediate test.

Revoked. We test if the TLS proxy accepts revoked certificates using Gibson Research Corporation's special site that hosts a website using a revoked certificate [10]. Digicert provided them with an intentionally revoked certificate using both a Certificate Revocation List (CRL) and the Online Certificate Status Protocol (OCSP). If the revoked certificate is allowed, this implies that the TLS proxy does not validate the revocation status of the delivered certificates and their appropriate issuers.

Expired and Not Yet Valid Certificates. We generate three distinct tests to check the behavior of network appliances when exposed to expired certificate. For the first test, we craft a root CA certificate with an expired validity date, and use it to sign a regular leaf certificate. For the second test, we craft a regular root CA certificate, use it to sign an expired intermediate certificate, which in turn, signs a regular leaf certificate. For the third test, we craft a regular root CA certificate, and use it to sign an expired leaf certificate. Similarly, we generate three similar tests for not yet valid certificates. The main difference between the two set of certificates is that, for expired certificates, the 'valid to' date is prior to the current date of testing, while the not yet valid certificates have the 'valid from' date exceeding the date of testing.

Invalid pathLenConstraint. A pathLenConstraint of 1 in a root CA certificate implies that the issuer can issue one layer of intermediate certificates, which in turn will have a pathLenConstraint of 0. A pathLenConstraint of 0 implies that this certificate can only issue leaf certificates. We generate a root CA certificate with a pathLenConstraint of 1, and issue an intermediate certificate with a pathLenConstraint of 0 using it, and subsequently issue another intermediate certificate with a pathLenConstraint of 0 using the first intermediate certificate. We then use the section intermediate certificate to issue a leaf certificate. Using this methodology, we test if the TLS proxies check the pathLenConstraint parameter, as the first intermediate should issue only leaf certificates, and not another intermediate certificate.

Bad Name Constraint Intermediate. We test if the TLS proxies validate the Name Constraint x509v3 certificate extension. We craft a regular CA certificate, and use it to sign an intermediate certificate that has a different domain than 'apache.host' solely permitted as a DNS name. We then issue a leaf certificate for the domain *apache.host* using that intermediate certificate. The validating proxy should typically terminate the TLS connection when exposed to such a case, as the intermediate certificate has an issuing permit constraint for a domain different than 'apache.host'.

Malformed X509v3 Extension Value. We generate a regular root CA certificate and use it to issue a leaf certificate that holds a dummy random string as a value for its keyUsage parameter (i.e., will not match any of the names in the list of the permitted key usages).

Unknown Critical X509v3 Extension. We generate a root CA certificate, and use it to issue a leaf certificate that holds a non-typical x509v3 extension (unusual OID value), set to critical. We thus analyze the TLS proxies' behavior when exposed to unknown extensions marked as critical.

Wrong keyUsage and extKeyUsage. We rely on two tests for the keyUsage and extKeyUsage x509v3 extensions, one for leaf certificates, and the other for root certificates. Regarding the keyUsage, we craft a regular root certificate, and sign a leaf certificate that holds a keyUsage value of keyCertSign, omitting the required keyEncipherment value for all leaf certificates. TLS proxies should drop TLS connection to servers that hold no keyEncipherment keyUsages. Moreover, we craft a root CA certificate with a keyUsage of nonRepudiation, omitting the required keyCertSign value for all issuing certificates. TLS proxies should not accept issuing certificates with a keyUsage value that excludes keyCertSign.

Regarding the extKeyUsage extension, we craft a regular root certificate and use it to issue a leaf certificate that holds clientAuth as the extKeyUsage value, implying that this certificate is meant to be used by TLS clients, and not by TLS servers. TLS proxies should then drop the TLS connection. We also craft a root certificate whose extKeyUsage value consists of codeSigning, implying that this issuing certificate is not meant to be used for TLS connections. TLS proxies should similarly drop such a connection. Failure of proper validation of certificate usages allows attackers to abuse TLS connections by using non-compliant certificates.

Short Key Length Root and Leaf Certificates. We generate RSA-512 and RSA-1024 root CA certificates, and import them to the trusted stores (when possible). We host their respective leaf certificates, and test if the TLS proxy accepts insecure key sizes for root certificates. On the other hand, we generate regular root certificates with proper key sizes (e.g., RSA-2048), and craft their leaf certificates to have short keys (512, 768, 1016 and 1024 as key sizes), and test if the TLS proxies accept such insecure key sizes for leaf certificates.

Bad Signature Hashing Algorithms. To check the proxies' behavior when exposed to weak and deprecated signature algorithms, we modify the signature algorithms in the OpenSSL configuration file when signing three distinct certificates to use respectively MD4, MD5 and SHA1.

B PRIVATE KEY EXTRACTION FOR CISCO IRONPORT WSA

In this section, we discuss the challenges faced while attempting to extract the private key from the Cisco Ironport Web Security Appliance.

We performed several attempts to bypass the limited custom command line interface, and access the filesystem content itself. We first tried to skip the proprietary command-line interface and reach operating system's native command-line. However, Cisco's interface is designed to have no escape point out of its custom command-line interface [3]. We then attempted to mount the network appliance's drive to our Ubuntu machine. We discovered that the virtual disk drive is divided into 9 different partitions, with FreeBSD as the main OS. We subsequently attempted to mount all partitions, with FreeBSD's filesystem type UFS, and all UFS type options, which include: old, default, 44bsd, ufs2, 5xbsd, sun, sunx86, hp, nextstep, nextstep-cd, and openstep [25].

With all the mentioned UFS types failing and no mounted drive, we then attempted to explore the content of the disk drive without mounting it. We relied on the Linux *strings* command, which extracts printable characters from binary files. We then parsed the output and saved all private keys found, by searching for the private key delimiters '`-----BEGIN PRIVATE KEY-----`' and '`-----BEGIN RSA PRIVATE KEY-----`'. We then compared the modulus of each key to the appliance's public key certificate, in order to attempt to locate the corresponding private key.

With no positive matching, we proceeded to memory analysis. We dumped the volatile memory of a VM by saving a snapshot of the running machine, after intercepting the traffic for a not previously visited website. We ensured that the website visited has not been visited and proxied earlier, to guarantee that the private key will be used to sign the intercepted page, and thus, be located in the appliance's RAM. Subsequently, we passed the memory dump to Volatility, a memory forensics tool [29]. Volatility requires as an input the exact profile of the OS corresponding to the memory dump. Consequently, we attempted to determine the profile using Volatility's 'imageinfo' command, which fails to determine the profile. Without the specific profile, Volatility fails to execute.

As a result, we attempted to use the collected memory dump with Heartleech [11]. We fed the tool with Cisco Ironport Web Security Appliance's memory dump, along with the TLS proxy's public key certificate to Heartleech, which successfully outputs the corresponding private key.

C KNOWN TLS ATTACKS

Below we provide a brief description of the known TLS attacks mentioned in this work.

The BEAST [40] vulnerability allows an active MITM attacker to decrypt session cookies and authentication credentials. This client-side vulnerability is the result of a flaw in the CBC mode of operation, which can be exploited in TLS version 1.0 connections, by repeatedly predicting the Initialization Vector blocks and checking if the prediction is correct. Countermeasures to this vulnerability include: excluding CBC-mode ciphersuites, implementing the 1/n-1 split method [1] for CBC, or denying access to web servers that do not accept versions higher than TLS 1.0.

CRIME [41] is exploited when TLS data compression is used, leaking the content of some encrypted packets, including authentication cookies. To mitigate this attack, TLS data compression should be avoided.

A system is vulnerable to the FREAK [34] attack if it includes export ciphers in the list of supported ciphers. The use of these ciphers was forced by the US government during the 1990s for any system that is exported outside the US. Modern TLS implementations must not include export-grade ciphers into their list of supported ciphers.

The Logjam [32] attack is a reminiscent of the FREAK attack, requiring the use of the previously mentioned export ciphers. However, this attack exploits a vulnerability in the TLS protocol's DHE key exchange. The MITM attacker forces a DHE_EXPORT cipher with the web server, and relies on weak TLS parameters and a set of precomputed parameters to break the DHE key exchange process, resulting in decrypted traffic. To avoid this attack, TLS proxies should not advertise export-grade ciphers in the list of their supported ciphers.

TLS renegotiation [50] is a TLS protocol feature that permits renegotiation of new TLS parameters in an already established TLS connection. This feature can be exploited by an attacker, allowing the attacker to place herself in an MITM position between the requesting client and the origin server. This is achieved by establishing a TLS connection with the server, and later, intercepting the client's TLS request and relays the the request over the encrypted channel. The renegotiation of the parameters then takes place, without the consent of the client. On the other hand, the server believes that it is a legitimate TLS renegotiation. As a result, the attacker can now send requests to the server on behalf of the authenticated client. As a mitigation, TLS proxies must implement RFC 5746, a TLS extension that ties renegotiations to the initial connection.