Killed by Proxy: Analyzing Client-end TLS Interception Software

Xavier de Carné de Carnavalet and Mohammad Mannan
Concordia Institute for Information Systems Engineering
Concordia University, Montreal, Canada
{x_decarn, mmannan}@ciise.concordia.ca

***Abstract*—To filter SSL/TLS-protected traffic, some antivirus and parental-control applications interpose a TLS proxy in the middle of the host's communications. We set out to analyze such proxies as there are known problems in other (more matured) TLS processing engines, such as browsers and common TLS libraries. Compared to regular proxies, client-end TLS proxies impose several unique constraints, and must be analyzed for additional attack vectors; e.g., proxies may trust their own root certificates for externally-delivered content and rely on a custom trusted CA store (bypassing OS/browser stores). Covering existing and new attack vectors, we design an integrated framework to analyze such client-end TLS proxies. Using the framework, we perform a thorough analysis of eight antivirus and four parental-control applications for Windows that act as TLS proxies, along with two additional products that only import a root certificate. Our systematic analysis uncovered that several of these tools severely affect TLS security on their host machines. In particular, we found that four products are vulnerable to full server impersonation under an active man-in-the-middle (MITM) attack out-of-the-box, and two more if TLS filtering is enabled. Several of these tools also mislead browsers into believing that a TLS connection is more secure than it actually is, by e.g., artificially upgrading a server's TLS version at the client. Our work is intended to highlight new risks introduced by TLS interception tools, which are possibly used by millions of users.**

I. Introduction

Several antivirus and parental control software tools analyze client-end traffic, including HTTPS traffic, before it reaches browsers for reasons including: eliminating drive-by downloads, removing unwanted advertisements, protecting children's online activities by blocking access to unwanted websites, or simply hiding swear words. Such tools are possibly used by millions of users (cf. [30]); sometimes they are installed by OEMs on new computers (perhaps unbeknownst to the user), often downloaded/purchased by users, and after installation, remain active by default (although may not always perform filtering).

To analyze encrypted traffic, these tools generally insert an active man-in-the-middle (MITM) proxy to split the browser-to-web server encrypted connection into two parts: browser-to-proxy and proxy-to-web server. First, such a tool grants itself signing authority over any TLS certificate by importing its own root certificate into the client's trusted CA stores. Then, when a TLS connection is initiated by a client application (e.g., browser, email client) to a remote server, the TLS proxy forges a certificate for that server to "impersonate" it in the protocol. Client encryption effectively terminates at the proxy, which dutifully forms a second TLS connection to the remote server. The proxy inspects messages between the two connections, and forwards, blocks or modifies traffic as deemed appropriate. However, the use of such a proxy may weaken TLS security in several ways.

First, if the proxy's root certificate is pre-generated (i.e., fixed across different installations), users could be vulnerable to impersonation by an active MITM network adversary, having access to the signing key, if the proxy accepts external site certificates issued by its own root certificate; see Fig. 1. In Feb. 2015, the advertisement-inserting tool SuperFish [5] was found to be vulnerable to such an attack due to its use of the Komodia SDK, which pre-generates a single root certificate per product. As this SDK is used by other products, independent work tracked their root certificates and associated private keys.[1] In Nov. 2015, two Dell laptop models were found to be shipped with the same root certificate along with its private key [21]. The same attack is also possible, if the private signing key of a per-installation root certificate can be accessed by unprivileged malware in a targeted machine. Note that, unlike advertisement-related products, removing antivirus and parental control tools may not be feasible or desirable.

Second, as the TLS proxy itself connects to the server, it is in charge of the certificate validation process, which may be vulnerable to several known problems, including: accepting *any* certificate (cf. Privdog [15]), failing to verify the certificate chain, relying on an outdated list of trusted CAs, or failing to check revocation status. Brubaker et al. [12] show that certificate validation is a particularly error-prone task, even for well-known and tested TLS libraries and clients.

Third, the TLS proxy introduces a new TLS client (w.r.t. the remote server) in the end-to-end client-server connection. Similar to browsers, these proxies must be kept updated with the latest patches as developed against newly discovered vulnerabilities (e.g., BEAST [20], CRIME [55], POODLE [41], FREAK [9], and Logjam [1]). Outdated proxies may also lack support for safe protocol versions and cipher suites, undermining the significant effort spent on securing web browsers.

Permission to freely reproduce all or part of this paper for noncommercial purposes is granted provided that copies bear this notice and the full citation on the first page. Reproduction for commercial purposes is strictly prohibited without the prior written consent of the Internet Society, the first-named author (for reproduction of an entire paper only), and the author's employer if the paper was prepared within the scope of employment.
NDSS '16, 21-24 February 2016, San Diego, CA, USA
Copyright 2016 Internet Society, ISBN 1-891562-41-X
http://dx.doi.org/10.14722/ndss.2016.23374

[1] https://gist.github.com/Wack0/17c56b77a90073be81d3

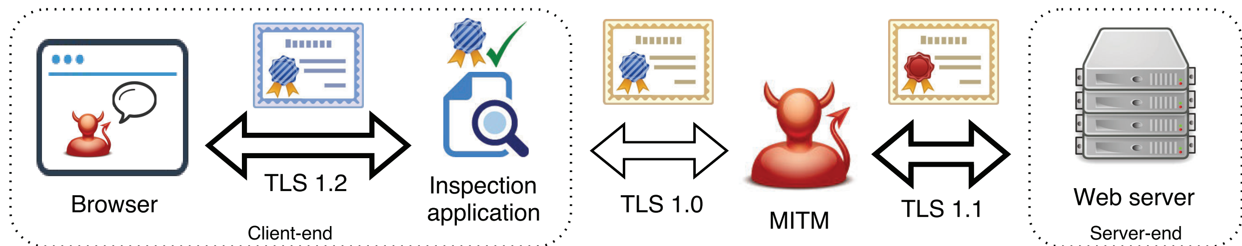

Fig. 1. Illustration of a man-in-the-middle (MITM) attack against a content-control application performing TLS interception that accepts its own root certificate as the issuer of externally-delivered certificates. In addition, TLS parameters are not transparent to browsers, and may be lowered by the proxy to an unwanted level. All SSL/TLS versions shown are the highest ones that can be negotiated between two parties, assuming the MITM supports at most TLS 1.2.

Fourth, the proxy may not faithfully reproduce a connection to the browser with the same parameters as the proxy's connection to the server. For example, the proxy may not match the use of extended validation (EV) certificates, and mislead the browser to believe that the connection uses lower or higher standards than it actually does; hence, the proxy may trigger unnecessary security warnings or suppress the critical ones. We refer to the capacity of a TLS proxy to reflect TLS parameters between both ends as proxy transparency (not to be confused with Certificate Transparency [24]).

Graham [26] shows how easy it is to retrieve the private key for SuperFish, and consequently to eavesdrop communications from clients using SuperFish in specific Lenovo laptops. Recently, Böck [11] listed several observations about three antiviruses, including vulnerability to CRIME and FREAK attacks, and the use of old SSL/TLS versions. Other studies (e.g., [16], [19]) also highlight the possible dangers of filtering by dedicated TLS interception appliances, targeted for enterprise environments.

In this work, we present a framework to analyze client-end TLS proxies, and report our results on 14 well-known antivirus and parental control tools for Windows (including two from the same vendor, and sometimes multiple versions), tested between March and August 2015. Analyzing these proxies poses additional challenges compared to testing regular clients (e.g., browsers), servers (e.g., HTTPS web servers), or standalone enterprise proxy appliances. Such challenges include: the lack of Server Name Indication (SNI) support (requiring one IP address per test) and filtering on specific ports only, both of which limit the applicability of existing online TLS test-suites; and difficulties to make a proxy trust our test root certificate due to the use of custom CA trusted stores (often encrypted/obfuscated in an undocumented manner). Following the structure of a TLS proxy, we use the framework to analyze client proxies from four perspectives: (a) root certificates of proxies, and protections of corresponding private keys; (b) certificate validation; (c) server-end TLS parameters; and (d) client-end transparency.

We found that *all* the analyzed products in some way weaken TLS security on their host. Three of the four parental control applications we analyzed are vulnerable to server impersonation because they either import a pre-generated certificate into the OS/browser trusted stores during installation, lack any certificate validation, or trust a root certificate "for testing purpose only" with a factorable 512-bit RSA key. The remaining one imports a pre-generated certificate when filtering is enabled for the first time, and never removes it even after uninstalling the product, leaving the host perpetually vulnerable. One antivirus did not validate any certificate in the first version we analyzed, then changed to prompting the user for each and every certificate presented on email ports (secure POP3, IMAP and SMTP), leaving users unprotected or in charge of critical security decisions. Another antivirus fails to verify the certificate signatures, allowing a trivial MITM attack when filtering is enabled. A third antivirus leaves its host vulnerable to server impersonation under a trivial MITM attack after the product license is expired (accepts all certificates, valid or otherwise). Due to the expired license, this product also cannot be automatically updated to a newer version that fixes the vulnerability. We contacted the affected companies and report their responses.

Finally, our framework can be applied to client-end proxies for Mac and on mobile platforms, found e.g., in Mobile Device Management (MDM) solutions. Also, as an integrated framework, it can guide more comprehensive testing of other TLS proxies, such as network appliances in business organizations used to ensure compliance with policies, e.g., US Health Insurance Portability and Accountability Act (HIPAA).

Contributions.

1) We design a hybrid TLS testing framework for client-end TLS proxy applications, combining our own certificate validation tests with tests that can be reliably performed through existing test suites (see Section V). Using this framework, we analyzed 14 leading antivirus and parental control products under Windows that offer HTTPS/secure email filtering, or at least install a root certificate in the client's trusted CA stores (OS/browsers) to expose potential TLS-related weaknesses introduced by these tools to their hosting systems.
2) We investigate whether the tools generate product-specific root certificates dynamically, and to what extent they protect the associated private keys. We perform an extensive analysis of certain products to recover their private keys, requiring non-trivial reverse-engineering and deobfuscation efforts (although one-time only, for each product). When the same key is used on all systems using the same product, simple MITM attacks are possible (see Section III).
3) We expose flaws in the certificate validation process of the TLS proxies, given only a small corpus of carefully-crafted invalid certificates, which include expired and revoked certificates along with chains of trust that are broken for various reasons (see Section VI). While testing our invalid certificates, we faced several challenges that are not generally considered in existing client TLS tests (cf. Qualys [52] and others [10], [64]; see Section IV).
4) We analyze the TLS proxies against known attacks, and test their support for the latest and older TLS versions.

We also test whether the TLS version negotiated with the server differs from what the browser sees (as supplied by the proxy), along with various other parameters, e.g., certificate key size, signature hashing algorithm, EV certificates. We observe that browsers (and in turn, users) are often misled by these proxies (see Section VI).

5) We discuss implications of our findings in terms of efforts required for launching practical attacks (see Section VII), and outline a few preliminary suggestions for safer TLS proxying (see Section VIII).

II. Background and threat model

In this section, we provide details of our product selection, terminologies and threat model as used in this paper.

A. Terminologies

We refer to content-control applications as CCAs, or simply products; these include antivirus and parental control applications when they perform some form of traffic filtering. Products that support TLS filtering are termed as TLS proxies, or simply proxies. Each product imports a root certificate in the OS trusted CA store for the proper functioning of their proxy, and possibly other third-party stores (primarily browser CA stores).

A proxy acts between a client application and a remote server. Client applications include web browsers, email clients, OS services, and any other TLS clients. We mostly discuss the consequences of bad TLS proxies from a browser's perspectives, considering browsers as the most critical TLS client application for users; however, other applications/services may also be affected. We use the terms browsers and client applications interchangeably. For browsers, we consider Microsoft Internet Explorer (IE), Mozilla Firefox and Google Chrome.

B. Product selection

We relied on AV-comparatives.org [6], [7], Wikipedia[2] and other comparatives [65] to select well-known antivirus and client-end parental control products under Windows. When a vendor offers multiple versions of an antivirus or network firewall, we review the specifications of each product to find the simplest or cheapest one that supports TLS/HTTPS interception; if the specifications are unclear, we try several versions. Our preliminary test-set includes a total of 55 products (see Table V in Appendix D): 37 antiviruses and 18 parental control applications. Fourteen of these tools import their own root certificates in the OS/browser trusted CA stores, and 12 of them actually proxy TLS traffic. The rest of our analysis focuses on these 14 applications/12 proxies. Several of these proxies have also been identified as a major source of real-world traffic filtering (see e.g., [30], [51]).

C. Insertions in trusted stores: implications

There are several trusted stores that can be affected by CCAs. Windows provides a trusted store that we refer to as the OS trusted CA store, while third-party applications may maintain their own store (e.g., Mozilla Firefox, Opera); see Appendix A. CCAs install a root certificate in a trusted store so that TLS applications relying on that store accept TLS connections filtered by the proxy without any warning or error. However, an imported CCA root certificate implies that those TLS applications thereafter automatically trust *any* web content signed by that certificate, not simply the filtered content. When the CCA is manually disabled or uninstalled, or the CCA stops filtering due to an expired license, the root certificate may still remain in the trusted store. Also, we observed that these CCA root certificates are valid for a period of one to 20 years (11 out of 14 are valid for 10 years). As a consequence, TLS clients may be vulnerable to impersonation attacks when the private key for the root certificate is not suitably protected. Example scenarios include: CCAs that simply reuse the same public/private key pair across installations; CCAs that do not remove a root certificate from the trusted stores and the corresponding private key becomes compromised later (e.g., a RSA-1024 root certificate valid for 10 years leaves plenty of time for a dedicated attacker to factor the key). Compared to installing a new application, inserting a root certificate in a trusted store has more security implications that may span even beyond the product's lifespan. Such insertions are also mostly invisible to users, i.e., no explicit message is displayed by the OS, CCAs, or browsers, beyond granting generic admin privileges to the CCAs.

D. Threat model

To exploit the vulnerabilities identified in our analysis, we primarily consider two types of attacks (see below). In both cases, we assume an attacker can perform an active MITM attack on the target (e.g., an ISP, a public WiFi operator), and the goal is to impersonate a server in a TLS connection, or at least extract authentication cookies from a TLS session. Attackers cannot run privileged malware (e.g., rootkits) in a target system, as such malware can easily defeat any end-to-end encryption. However, attackers can execute privileged code in their own machines to study the target products.

Generic MITM: The attacker may learn (e.g., from network access log) whether a vulnerable CCA is installed on a target system; otherwise, a generic MITM attack can be launched against all users in the network, with the risk of being detected by users who are not vulnerable. Typically, CCAs that install pre-generated certificates may enable such a powerful attack, if the corresponding private keys can be retrieved (on an attacker controlled machine). No malicious code needs to be executed on the target system.

Targeted MITM: The attacker can run unprivileged code on the target system, prior to the attack (e.g., via drive-by-downloads, social engineering). Such malicious code can extract a dynamic, proxy-generated private key, which can then be used to impersonate any server at that specific target system.

III. Private key extraction

Most CCAs implement various protection mechanisms to safeguard their private keys on-disk. In this section, we discuss our methodologies to identify the types of protection as used by CCAs, and how we extract plaintext private keys from application-protected storage. OS-protected private key extraction requires admin privileges, excluded in our threat model for targeted attacks (see Appendix B).

Overview. Our primary goal here is to extract private keys from disk on a user's machine, using only unprivileged code. Extracting private keys from memory requires admin privileges, and we consider such an approach for two cases: to extract private keys associated to pre-generated certificates,

[2] https://en.wikipedia.org/wiki/Comparison_of_antivirus_software, and /wiki/Comparison_of_content-control_software_and_providers

and to understand the application process dealing with an in-memory private key to identify how the key is stored/protected on disk. We discuss the protection mechanisms used by our tested CCAs; we circumvented the two main on-disk protection mechanisms without requiring admin privileges on the target system. We then discuss some contextual security aspects.

A. Locating private keys in files and Windows registry

Most CCAs (optionally generate and) import their root certificates into OS/browser trusted stores during installation. Using Process Monitor ("procmon" from Microsoft/SysInternals), we monitor all the application processes of a CCA during installation. After installation, we manually check for any newly added trusted CA using the Windows Certificate Manager. If a new entry in the Windows store is inserted, searching for the SHA1 fingerprint of that certificate in procmon's log identifies the exact event where the entry was created. We can thus identify the specific application process that inserted the new certificate, and possibly identify other affected files and registry locations, and which may potentially contain the associated private key. Specifically, we perform manual analysis (e.g., searching for keywords such as "certificate") on file and registry operations (potentially hundreds), executed right before and after the root certificate insertion. When a CCA leverages the Windows CAPI/CNG, we find obvious traces in the log; and we can then easily identify the correct key in a protected container with a label that is often similar to the CCA's name.

We also explore a CCA's installation directory for files that appear to be certificates or keys (with extensions such as .cer, .crt, .cert, .pem, .key; or filenames containing *cert* or *CA*). If a private key is found, we match it to the root certificate for confirmation. We also check whether the key file is accessible by unprivileged code, allowing targeted MITM attacks.

If no root certificate is imported during installation, we explore the application's settings for the availability of TLS filtering, and enable filtering when found. We then reboot the system (sometimes required to activate filtering), and visit an HTTPS website in a browser to trigger TLS interception, forcing the proxy to access its private key. At this point, if no root certificate is installed and no sample HTTPS connections are filtered, we discard the application from the rest of our analysis. In the end, we fully analyze 14 products that support filtering and/or import a root certificate in the OS trusted store.

B. Application-protected private keys

Instead of using the OS-protected key storage, some CCAs store their private keys protected by the application itself, using encryption and sometimes additional obfuscation. After locating the on-disk protected private keys (Section III-A), we try to defeat such custom protections to extract the keys. Here, we detail our methodology to bypass two main protection mechanisms we encountered, requiring some reverse-engineering effort (non-trivial, but one-time only for each mechanism).

1) Identify the process responsible for TLS filtering: First, we find the application process responsible for handling a private key, and then investigate the corresponding binary files (DLLs) involved in this process to extract the passphrase/key used in encrypting the private key. As the private key must be in memory when a proxy is performing TLS filtering, we can identify the specific process responsible for filtering as follows: (a) Identify all the running processes of a target CCA, by finding services with related names or identifying new running processes following the CCA installation; (b) Dump the process memory of each of these processes; (c) Search the memory dumps for a private key that matches the root certificate's public key; and (d) Identify the process that handles the TLS filtering, i.e., the one that holds the private key in its memory space. As all CCAs in our study use RSA key pairs, and those that do not rely on OS-provided key storage use the OpenSSL library for handling keys, we use the heartleech tool [27] to search for a private key in the memory dumps, by specifying the corresponding root certificate.

2) Retrieving passphrases: We discuss three techniques used to extract a passphrase or the derived encryption key, to recover a target private key from an on-disk encrypted/obfuscated container. When a specific method is successful against a given CCA, it yields a static "secret" that allows for decryption of the private key using unprivileged operations, satisfying our threat model for targeted MITM attacks (see Section II-D).

Method 1: Extracting strings. We extract strings of printable characters from the binaries of the TLS filtering process, and use them as candidate passphrases. This method was used to recover the SuperFish private key (cf. Graham [26]).

Method 2: Disassembling/Decompiling. We disassemble the process binaries using IDA Pro, and search for selected OpenSSL functions related to private keys; we label such functions as passphrase consumer functions.[3] Then, we follow the source of the argument representing a passphrase, and locate potentially hardcoded passphrases. This method is quite effective as all tested CCAs use the OpenSSL library for private key operations, and IDA FLIRT can reliably identify such OpenSSL functions from process binaries.

Method 3: Execution tracing. Some CCAs may obfuscate a hardcoded encryption passphrase/key by performing additional computation on it, prior to calling a consumer function. These computations may not be accurately disassembled by IDA Pro, due to e.g., the use of ad-hoc calling conventions. In such cases, we rely on execution tracing. However, instead of debugging a live proxy process, we trace only selected parts from a proxy, by executing those parts independently.[4] We first load a candidate binary containing consumer functions into a debugger (Immunity Debugger[5] in our case), and set breakpoints on these functions. Then, we change the binary's entry point to a function that is two/three function calls away from a consumer function, as we do not know the precise location of instructions processing the passphrase/key. Using this method, we identified all remaining runtime-generated passphrases that could not be extracted through Methods 1 and 2. Note that if the encryption key is dynamically generated from runtime parameters (as opposed to hardcoded), further reverse-engineering is needed to extract the logic to generate the correct key on a target machine. In practice, we only encountered static encryption keys.

[3] Examples: SSL_CTX_use_PrivateKey, SSL_CTX_use_PrivateKey_file, PEM_write_RSAPrivateKey, X509_check_private_key, PKCS8_decrypt.

[4] Debugging a live proxy is complicated by several factors: a proxy often operates as a Windows service, requiring kernel-level debugging; services are often started early in the boot process and may access the private key before we can debug the execution; services may not be restarted afterwards without rebooting; and services may use anti-debugging techniques.

[5] http://immunityinc.com/products/debugger/index.html

3) Encrypted containers: Some CCAs protect on-disk private keys using encrypted database containers such as SQLCipher, an extension of SQLite with AES-256 encryption support. While techniques from Section III-B2 are mostly effective against SQLCipher, we develop a generic method that can possibly be used with any encrypted SQLite variant. This method helped us unlock an encrypted container that uses a modified version of SQLCipher. We locate SQL queries in the target binary that are executed immediately after the database is opened. By modifying such a query to *PRAGMA rekey=‘’*, we instruct the SQL engine to reencrypt the database with an empty key, essentially decrypting the database containing the intended private key. When we need to make a CCA operate with our decrypted/modified database, we also patch the CCA's binary not to require a passphrase when opening the database. This is particularly useful for CCAs relying on their own trusted stores saved within a SQLCipher database, which we must modify to insert our test root certificate (see Section V-C).

C. Security considerations

When the private key corresponding to a proxy's root certificate is retrieved, new security considerations emerge, as discussed below; a proxy must be tested accordingly.

Time of generation. Some CCAs come with a preloaded root certificate that they import during installation or when TLS filtering is activated. We label such certificates as pre-generated, which may enable generic MITM attacks. In contrast, others may generate a fresh root certificate unique to the local machine; we label such certificates as install-time generated. If the private key of an install-time generated certificate is accessible from unprivileged code, a targeted MITM attack becomes possible. We verify whether a certificate is generated at install-time or pre-generated by simply installing the product on two different machines with distinct environments (e.g., different hardware, x86 vs. x86-64), and compare the installed certificates. We also search for pre-generated certificate files and private keys in the installer.

Entropy during generation. It is possible that the entropy used during the generation of a new public/private key pair in install-time generated certificates is inadequate. In practice, since most products we analyzed generate a root certificate with RSA keys using OpenSSL, the generation process is expected to call certain known functions, e.g., `RAND_seed()`, `RAND_event()`, `RSA_generate_key_ex()`; we found calls to the last function in many cases. However, we did not investigate further the key generation algorithm in CCAs.

Self-acceptance. For TLS interception, there is no need for a TLS proxy to accept proxy-signed remote certificates, as the proxy's root certificate is intended only to be used in the local machine. A proxy must not accept such remote certificates; otherwise, it becomes vulnerable to generic (for pre-generated root certificates), or targeted (for install-time generated root certificates) MITM attacks that use a forged certificate, signed by the proxy's private key.

Filtering conditions. CCAs may only filter TLS traffic under specific conditions. For example, filtering may be activated by default after installation, or offered as an optional feature disabled by default. Filtering may be applied only for selected categories of websites (especially for parental control tools), or for all websites. Filtering could also be port-dependent, or applied to any TCP port. Finally, only specific browsers/applications may be filtered. Self-acceptance is only relevant when the proxy is actively filtering. It may happen that the proxy is not enabled by default; however its root certificate is already imported in trusted stores.

Expired product licenses. CCAs may stop filtering traffic when their license or trial period is expired. If a proxy's root certificate is still present in trusted stores, it leaves browsers vulnerable to potential generic or targeted MITM attacks. This is especially relevant if the TLS proxy does not accept its own root certificate as a valid issuer for site certificates before license expiration; i.e., users are not vulnerable to MITM attacks involving a proxy-signed certificate before license expiration but become vulnerable afterwards. Alternatively, a CCA may decide to continue filtering traffic even in an expired state. In this case, we test whether the proxy's certificate validation process is still functional (e.g., rejects invalid certificates).

Uninstallation. When a CCA is uninstalled, its root certificate should be removed from OS/browser trusted stores. Otherwise, it may continue to expose browsers to MITM attacks, e.g., if the certificate is pre-generated, or the private key of an install-time generated certificate has previously been compromised.

IV. Limitations of existing TLS test suites

Existing test suites possess certain limitations that prevent them from being used directly to test client-end TLS proxies. Note that such test suites have not been designed for the TLS proxies we target. We summarize these limitations below, and address them in our framework.

A. Certificate verification

After the Komodia incident [5], to check whether users are affected by Komodia-based interception tools, several web-based test sites appeared (e.g., [67], [10]). These tests are based on loading a CSS or JavaScript file hosted on a server with an invalid certificate (e.g., signed by the pre-generated root certificate of a broken TLS interception tool). If the CSS/JavaScript resource is successfully fetched, the client is then notified about the vulnerability. To test client-end TLS proxies, the following limitations must be addressed.

Unimplemented SNI extension. Certificate validation tests are often served on subdomains that are hosted from the same IP address since it is usually costly to use a unique IPv4 address per test. To distinguish multiple domain names, the server implicitly relies on the Server Name Indication (SNI) TLS extension to receive the hostname requested by the client at connection time. SNI has been widely adopted in modern browsers and TLS clients [18]. However, we encountered a few proxies that use ad-hoc ways to relay a TLS connection to the real server, without using the SNI extension. Test servers are thus unable to properly identify the requested host and are forced to deliver a default certificate, and eventually a 4xx error. For example, while badcert-superfish.tlsfun.de delivers a certificate signed by SuperFish's pre-generated certificate when the SNI extension is used, lacking SNI results in a 400 Bad Request webpage owned by the hosting company, served under their own domain name's certificate. Thus, the test would report that a carefully-crafted invalid certificate was not accepted (i.e., the proxy is not vulnerable), while the real reason is due to the wrong domain name. As a result, the invalid certificate is never tested against the proxy.

Caching-incompatible. A TLS proxy may cache certificates as seen from an initial connection to a server and reuse

them upon further visits to the same website. Some suites are apparently incompatible with caching proxies, especially when numerous certificates must be tested (e.g., Frankencert [12] uses 8,127,600 test certificates presented on *localhost*).

Undetected passthrough. Certain proxies only filter selected connections, e.g., only specific categories of websites or supported TLS versions; other connections are simply forwarded to a browser, letting the browser to deal with untrusted certificates or unsupported configurations. To test whether a proxy trusts its own root certificate, we must verify that content delivered by a web server with a proxy-signed certificate is successfully inspected. If the proxy chooses to passthrough this connection, the browser will simply accept the proxy-signed certificate (as if the proxy has generated the certificate as part of an active filtering process). We must make sure that the proxy was trying to filter the connection, and that it detected its own root certificate as the issuer, or simply did not find the issuer in its trusted store, and decided to let the browser deal with an untrusted issuer error. When successfully inspecting the connection, the proxy re-generates a similar certificate on-the-fly with a different key. Hence, the certificate received by the browser must be verified, e.g., by its fingerprint.

Fragile implementations. Proxies may behave inconsistently in specific test cases, leading to nondeterministic test results. For example, if several simultaneous connections are attempted to web servers with invalid certificates, a proxy may crash, or deny all future connections. Even a simple invalid certificate could lead to timeouts and incorrect test outcomes. Special care must be taken to test such buggy proxies.

Client-dependent filtering. Proxies may filter or accept only specific clients; e.g., while common browsers are filtered, we found that the OpenSSL toolkit launched from the command line was not filtered by half of the proxies. Sometimes, only selected browsers are filtered. This restriction is implemented simply by checking process names, or through a more involving mechanism (e.g., using non-obvious program characteristics). Thus, a proxy-testing client application must make sure that its connections are processed by the proxy.

B. *TLS security parameters*

Existing test suites, e.g., Qualys [52] and howsmyssl.com, perform an extensive test of TLS parameters (and relevant features), including: protocol versions, cipher suites, TLS compression, and secure renegotiation. Various sites also evaluate high-impact vulnerabilities; e.g., freakattack.com for the FREAK attack and weakdh.org for Logjam. As TLS parameters are generally tied to a server rather than a domain, online test suites resort to serving these tests on several TCP ports (e.g., [52], [64]). However, this solution is inadequate, as CCAs generally filter only specific ports (e.g., 80 and 443), sometimes non-configurable. We also found an antivirus that only analyzes encrypted emails on ports 465, 993 and 995. Thus, existing sites cannot properly test these TLS proxies.

V. Our TLS proxy testing framework

We design a hybrid solution combining our own certificate validation tests with tests that can be reliably performed through existing test suites. We discuss our methodology for testing certificate validation engines of the proxies, TLS parameters as apparent to browsers and remote servers, and known TLS attacks against each proxy.

A. *Test environment*

We setup a target TLS proxy in a virtual machine running Windows 7 SP1, and a test web server in the host OS. To address the lack of SNI support in proxies, we assign multiple IP addresses to a single network interface to map various test domain names to different IP addresses. We also instrument a DNS server on the host to serve predefined IP addresses in response to a query for our test domain names. For example, we map wrong-cn.local.test to 192.168.80.10, assign this IP to the network interface, and configure the web server to serve the corresponding certificate with a wrong CN field for requests made to that IP address. While private IPv4 address spaces can assign up to 16,387,064 individual addresses (far enough to map all our tests), a few CCAs do not to filter traffic from these address spaces. Thus, we also configure our test environment to use Internet-addressable IPs from a randomly picked range.

If all ports are filtered by the target TLS proxy (or ports are configurable), we simply leverage existing online testing suites to analyze the proxy for security-sensitive TLS parameters. Otherwise, we use a TCP proxy on the host to forward traffic addressed to these test suites from a proxy filtered port to the real server port. In this setup, we must preserve the correct domain names to avoid HTTP 300 redirections. While testing the TLS proxy on multiple server ports, we effectively need to serve several tests through the same test IP and port of our TCP proxy. To avoid caching issues, we restart the VM (with the TLS proxy) after each test. Our testing environment is made to conduct all tests within a single physical machine, requiring the CCA to be installed within a VM. Alternatively, two physical machines could also be used.

B. *Certificate validation testing*

We generate test certificates signed by the private key corresponding to our root certificate; we also make the proxies to trust our root certificate (see Section V-C). We visit test web pages using a browser filtered by the proxy under test (preferably Chrome, since it relies on the OS trusted store and provides details about the main connection). We use a couple of valid, control certificates to verify that a TLS proxy accepts our root certificate, or does not perform any filtering in a given setting (e.g., an unfiltered IP range, domain name or TLS version). When filtering is active, we test each TLS proxy with 9 certificates with a broken chain of trust, including: self-signed certificate, signature mismatch, non-trusted authority with the same name as a valid authority, wrong domain name, unknown issuer, non-CA intermediate authority, X.509v1, revoked and expired certificates; see Appendix C.

We also examine whether the proxies accept certificates with deprecated algorithms (e.g., RSA-512 and MD5), or algorithms that are being gradually phased out (e.g., RSA-1024, SHA1).[6] Regarding proxy transparency of a certificate's extensions and parameters, we examine how the proxy deals with Extended-Validation (EV) certificates, and whether the key length and hashing algorithm in a proxy-signed certificate are identical to the original server certificate.

[6]Firefox 42.0 and Chrome 47.0 still accept RSA-1024 keys in leaf certificates (as of December 2015); however, the trust in CAs using 1024-bit keys is being progressively revoked [45]. The use of MD5 for certificate signature has also been banned by modern browsers during 2011 (e.g., [42]) due to obvious forgery attacks [60]. SHA1 is also gradually being phased out (e.g., [25]).

Our small corpus of 15 certificates is intended to identify the most obvious validation errors. More comprehensive analysis (cf. [12]) can be performed by identifying the TLS library and version used by a CCA, and running more tailored tests against the library. In practice, we observed that most CCAs rely on OpenSSL or Microsoft Secure Channel (Schannel); however, more reverse-engineering is needed to accurately report which library is effectively used as the TLS stack by a given CCA. Additional certificates can also be generated to test whether the proxies interfere with recent enhancements to TLS (e.g., key pinning, HSTS). Note that in Chrome 47 (the latest version, as of December 2015), key pinning is overridden when a local TLS proxy filters connections.[7]

C. Proxy-embedded trusted stores

To validate server certificates, proxies may rely on the OS trusted store, or on a custom embedded store. Below we discuss testing considerations related to such custom stores.

Trusting our own root certificate. A valid issuer is required for signing several of our test certificates (e.g., expired, wrong CN, weak keys, or testing TLS support); we sign such certificates with a well-formed X.509v3 root certificate we generated (with RSA-2048). We make the proxies trust our root certificate, when possible. Note that a valid wildcard certificate (issued by a real CA) is insufficient for our purpose. Rather, we require a certificate that can be used to issue additional certificates (i.e., similar to an intermediate CA certificate); at the end, we did not obtain such certificates from a real CA as we do not meet the eligibility requirements (e.g., being a middle/large organization with a substantial net worth).

Usually, it is sufficient to import our root certificate into the OS/browser trusted stores. However, several CCAs rely on their own embedded stores (sometimes obfuscated), effectively introducing a new independent trusted CA store without any documented policy (cf. Mozilla [43]). We tried to insert our certificate in the proxy-trusted stores (see Section III-B3).

If we cannot make a proxy trust our root certificate, we generate relevant test certificates using the proxy's root certificate (with its retrieved private key). However, not all proxies trust their own root certificates to sign arbitrary certificates (as expected). In such cases, we search for external web servers with similar certificates, and visit them to test the proxy. Since we do not control external test websites, there is a possibility that our local tests yield different results than the online ones. We still provide both methods as the local tests can be made more comprehensive while online tests can serve as a backup solution to test at least certain available cases.

For example, an expired certificate can be tested at expired.badssl.com, if the proxy supports SNI. A wrong CN can be tested thanks to misconfigured DNS entries (e.g., tv.eurosport.com pointing to Akamai's CDN servers, delivering a certificate for the CDN's domain name). For weak RSA keys and deprecated signature algorithms, we were unable to find online tests. This is an expected limitation, as valid CAs currently do not issue such certificates. Hence, these tests cannot be performed when the proxy does not trust its own root certificate or the root certificate we generate; we had one such proxy among our tested products.

Store analysis. We try to determine the provenance of proxy-embedded stores (if readable), and check for issues such as globally distrusted CAs (e.g., DigiNotar), expired CAs, and CAs with weak keys (below RSA 1024 bits). When we find expired CAs, we verify that the proxy correctly checks the period of validity of its trusted store by (a) importing our own expired root certificate into the store, (b) attempting to connect to a test page serving a valid certificate signed by that expired CA. If the page loads, the proxy introduces vulnerabilities through its custom store.

[7]https://www.chromium.org/Home/chromium-security/security-faq

D. TLS versions and known attacks

We test support for SSL 3.0, TLS 1.0, 1.1 and 1.2. We rely on Qualys to perform the version check, when a proxy's filtering is not port-specific. Otherwise, if we can generate a valid certificate for the proxy, using our own or the proxy's root certificate, we run an instance of the OpenSSL tool as a TLS server, configured to accept only specific versions of SSL/TLS on desired ports. Finally, if we cannot provide a valid certificate, we simply proxy traffic from a proxy-filtered port to the Qualys server's real port. Following this methodology, we can detect vulnerabilities to POODLE, CRIME and insecure renegotiation. We also check how TLS versions are mapped between a browser and the proxy, and the proxy and the remote server (cf. Fig. 1). Any discrepancy in mapping would mislead the browser into believing that the visited website offered better/worse security than it actually does. This problem is particularly important when SSL 3.0 connections are masqueraded as higher versions of TLS.

Browsers support an out-of-specification downgrade mechanism for compatibility with old/incompatible server implementations [41], [13]. When a browser attempts a connection and advertises a TLS version unsupported by the server (e.g., TLS 1.2 in the ClientHello message), a broken server implementation may simply close the connection. The browser may then iterate the process by presenting a lower TLS version (e.g., TLS 1.1). This mechanism can be abused by an active MITM attacker to downgrade the protocol version used in a TLS communication, while both parties actually support a higher version. Abusing this mechanism is at the core of the POODLE attack. We verified whether proxies also implement this behavior by simulating such a broken server implementation (by simply closing the connection after receiving ClientHello, and inspecting further ClientHello messages).

We then analyze the list of ciphers presented by the proxy to the remote server using Qualys and howsmyssl.com. Weak, export-grade and anonymous Diffie-Hellman (DH) ciphers can be detected by these tests. When supporting TLS 1.0 (or lower) and CBC-mode ciphers without implementing mitigations (cf. record splitting [61]), proxies are vulnerable to the BEAST attack [20]. howsmyssl.com allows to test this scenario only when a proxy does not support TLS 1.1 or 1.2. We patched howsmyssl [28] and deployed it locally to test for the remaining cases. If the TLS version is not made transparent by the proxy, the cipher suites cannot be transparent either. Finally, we verify the proxy's vulnerability to FREAK and Logjam attacks using freakattack.com and weakdh.org.

VI. Results analysis

In this section, we provide the results of our analysis of the CCAs we considered, using our framework. We uncover several flaws that can significantly undermine a host's TLS security; we discuss practical attacks in Section VII.

A. Root certificates

We discuss the results of 14 products (out of the 55 initially analyzed) that install a root certificate in the OS/browser trusted CA stores; see Table IV in Appendix for a summary.

1) Certificate generation: CYBERsitter and PC Pandora use pre-generated certificates; the remaining 12 CCAs use install-time generated certificates, two of which do not perform any TLS-filtering (BullGuard AntiVirus (AV) and ZoneAlarm). For ZoneAlarm, we could not find any option to enable TLS interception in its settings. Since its antivirus engine is based on the Kaspersky SDK, we could find a file tree structure similar to Kaspersky Antivirus. In particular, the files storing the root certificate along with its plaintext private key reside in similar locations in both cases. For ZoneAlarm, the certificate file is named after what seems to be an undefined variable name, "(fake)%PersonalRootCertificateName%.cer". Apparently, ZoneAlarm developers were unaware that the SDK generates and installs this root certificate (or chose to ignore it), readable from unprivileged processes.

Additionally, when activating ZoneAlarm's parental control feature, a rebranded version of Net Nanny is installed. We also separately analyze the original version of Net Nanny (an independent parental control application). In turn, this bundled Net Nanny installs a second (pre-generated) root certificate; however, we were unable to trigger TLS filtering.

2) Third-party trusted stores: Among third-party trusted stores, we only verify and report our results for Mozilla Firefox; other applications such as Opera (and Mozilla Thunderbird when CCAs also target emails) may have also been affected. Eight of the 14 CCAs import their root certificates in the Firefox trusted store.

3) Self-acceptance: From the 12 products that support filtering, BullGuard Internet Security (IS) and AVG do not accept certificates signed by its own root certificate. However, AVG lets browsers continue the communication without any filtering. The browser is then left to accept site certificates signed by the proxy's root certificate as if they were issued by the local proxy. Others happily trust any site certificate issued by their root certificates.

We searched all the certificates from a ZMap [22] scan on July 21, 2015[8] to find certificates issued by any of the 14 root certificates from our CCAs. Finding such certificates would indicate exploitation of proxies supporting self-acceptance. We found only one such certificate at a Russian hosting site (signed by the "Kaspersky Antivirus Personal Root Certificate").

4) Filtering conditions: Eight CCAs activate TLS filtering upon installation, four provide an option, and the two others perform no filtering. Six CCAs only filter traffic from/to specific browsers. PC Pandora disallows browsers other than IE by aborting connections. KinderGate only filters specific categories of websites by default (related to, e.g., advertisement, dating, forums, nudity, social networking). Finally, the March 2015 version of Kaspersky lacks certificate validation for at least a minute after Windows is started up.

5) Expired product licenses: The version of Kaspersky we analyzed in March 2015 continues to act as a TLS proxy when a 30-day trial period is expired; however, after the license expiration, it accepts *all* certificates, including the invalid ones. The August 2015 version corrected both issues; however, customers who installed the vulnerable product version and did not uninstall it, remain vulnerable to a generic MITM attack as they do not benefit from automatic updates that could solve the issues (since their license has expired). Other CCAs either disable their proxy after expiration, or continue filtering with similar validation capabilities as before.

6) Uninstallation: Eight CCAs do not remove their root certificates from the OS/browser trusted stores after uninstallation, leaving the system exposed to potential attacks.

[8]https://scans.io/series/443-https-tls-full_ipv4

TABLE I. PROTECTIONS FOR A ROOT CERTIFICATE'S PRIVATE KEY

	Location	Protection	Access
Avast	CAPI	Exportable key	Admin
AVG	Config file	Obfuscation	Unknown
BitDefender	DER file	Hardcoded passphrase	User
BullGuard AV	DER reg key	Hardcoded passphrase	User
BullGuard IS	DER reg key	Hardcoded passphrase	User
CYBERsitter	CER file	Plaintext	User
Dr. Web	CAPI-cert[1]	Exportable key	Admin
ESET	CAPI	Non-exportable key	Admin
G DATA	Registry	Obfuscated encryption	User
Kaspersky	DER file	Plaintext	User
KinderGate	CER file	Plaintext	User
Net Nanny	Database	Modified SQLCipher	User
PC Pandora	CAPI-cert	Non-exportable key	Admin
ZoneAlarm	DER file	Plaintext	User

[1] CAPI-cert means that the private key is associated with the certificate

B. Private key protections

We provide below the results of our analysis on retrieving protected private keys; see Table I for a summary. We also explain how we retrieved four passphrase-protected private keys and a key stored in a custom encrypted SQLCipher database; our mechanisms illustrate why such protections are unreliable (although require non-trivial effort to defeat).

Summary. CCAs store private keys as follows: plaintext (CYBERsitter, Kaspersky, KinderGate and ZoneAlarm); CAPI/CNG encrypted (Avast, Dr. Web, ESET and PC Pandora); and application encrypted (six applications). Out of the six application-encrypted private keys, we are able to decrypt five with our methodology from Section III-B2. AVG appears to store its private key in a custom configuration file with an obfuscated structure. The types of protection we encountered are static, i.e., the *secret* used to protect a private key is fixed across all installations, requiring only a one-time effort. The results here are reported for the latest versions of the CCAs (August 2015); some results are for March 2015 versions (explicitly stated).

1) Passphrase-protected private keys: BitDefender stores its private key protected by a simple hardcoded passphrase typically found in cracking dictionaries; we retrieved the passphrase using Method 1. G DATA also protects its private key stored in registry using a custom format and a random-looking hardcoded passphrase (Method 1). Using Method 2, we found that BullGuard AV/IS generate the final passphrase at runtime based on a hardcoded string, as a form of simple obfuscation. In all cases, the passphrases are fixed across installations, and the protected private keys are readable by unprivileged processes, enabling targeted MITM attacks as defined in Section II-D. We do not report the plaintext passphrases to avoid obvious misuse.

2) Encrypted containers: Net Nanny relies on a modified SQLCipher encrypted database to protect its settings (scattered in multiple database files), including its private key. We provide details on Net Nanny to highlight the challenges posed by cus-

tom obfuscation techniques, which can be defeated with some effort (i.e., achieve less protection than OS-protected keys).

We noticed that one of Net Nanny's DLLs (db.dll) exports a few functions with meaningful names, apparently relating to SQLite. Following some differences in the functions names with the official *sqlite3* project, we realized that the DLL actually uses *IcuSqlite3*.[9] A quick search revealed that the IcuSqlite3 developer apparently works for ContentWatch, the company developing Net Nanny. From this connection, we assumed that IcuSqlite3 was used in Net Nanny, which benefited us by complementing the disassembly of db.dll by IDA Pro.

We were able to extract Net Nanny's passphrase using Method 3, which contained the name of the developing company. We failed however to simply leverage SQLCipher to open the encrypted databases.[10] Using the method from Section III-B3, we could successfully decrypt the first two databases before the program crashed. We rotated the database files until all were decrypted, and then found Net Nanny's root certificate and private key in a database. In the March 2015 version, we found that the proxy was using a pre-generated certificate, which made it vulnerable to a generic MITM attack in its default configuration. In the August 2015 version, the private key is install-time generated. A targeted MITM attack is still possible (the databases are readable from unprivileged processes). Furthermore, the private key is passphrase-protected by a long random string, also stored in the database. We also made Net Nanny to trust our root certificate by inserting it in Net Nanny's custom root CA list, stored in the encrypted databases.

C. Certificate validation and trusted stores

Our certificate validation analysis reveals various flaws in nine out of 12 proxies.

1) Invalid chain of trust: We use nine test certificates with various errors in their chain of trust; see Table II. We highlight the dangerous behaviors in the table ("Accept" and "Changed"). If a proxy can detect a certificate error, it may react as follows: send the browser a certificate issued by an untrusted CA ("u-CA" in the table), typically named "untrusted" along with the proxy's name; send a self-signed certificate ("S-S"); ask confirmation from the user by delivering a warning webpage or an alert dialog ("Ask"); or, terminate the connection altogether ("Block"). For expired certificates, the period of validity may be passed as-is to the client ("Mapped"), or updated to reflect a working period ("Changed"); in the latter case, the browser cannot detect if the original certificate has expired. For certificates issued for the wrong domain name, the CN field may be passed as-is to the browser, or may be changed to the domain name expected by the browser. Finally, proxies may entirely fail to detect invalid certificates, exposing browsers to generic MITM attacks ("Accept").

Only Kaspersky and Net Nanny successfully detected all our invalid certificates; however, when detected, the user is asked to handle the error. In contrast, most browsers now make it significantly difficult to bypass such errors (e.g., complex overriding procedure), or simply refuse to connect. AVG also detected the 6 invalid certificates we tested. We could not perform the remaining tests on AVG, as it is immune to self-acceptance, and we could not make it trust our own root certificate; online tests were also inapplicable.

In contrast, CYBERsitter, KinderGate and PC Pandora accepted nearly all invalid certificates we presented. The March 2015 version of G DATA also accepted *all* certificates, while the August version requires user confirmation (via an alert window) for *all* certificates, including valid ones signed by legitimate CAs. BullGuard IS fails to validate the signature of a certificate, and accepts our signature mismatch and fake GeoTrust certificates. Apparently, BullGuard IS verifies the chain of trust only by the subject name, allowing trivial generic MITM attacks. Finally, we found that 9 proxies do not check for the revocation status of a certificate.

Proxy transparency. Validation errors such as wrong CN, self-signed, expired certificate, and unknown issuer, may cause modern browsers to notify users (and allow the connection when confirmed via complex UI); most proxies modify these errors, causing browsers to react differently. For example, BitDefender turns a wrong CN into a certificate signed by an unknown issuer, and CYBERsitter changes the CN field to make the certificate valid. Most other proxies relay the CN field as-is, or ask for user confirmation. Avast, AVG, BitDefender and Dr. Web change self-signed certificates to certificates issued by an untrusted CA. Conversely, BullGuard IS turns certificates signed by an unknown issuer into self-signed. The behavior for unknown CA, non-CA intermediate and X.509v1 intermediate is always identical for a given proxy, with the exception of Avast that blocks connections for the last two cases. Finally, we observed that all proxies but Avast filter HTTPS communications when the servers offer an EV certificate and present it as a DV certificate to browsers.

2) Weak and deprecated encryption/signing algorithms: We tested proxies against certificates using MD5 or SHA1 as the signature hashing algorithm, combined with weak (RSA-512) or soon-to-be-deprecated keys (RSA-1024). Nine out of 12 proxies accept MD5 and SHA1, implying that if an attacker can obtain a valid certificate using MD5 signed by any proxy-trusted CA, she can forge new certificates for any website (generic MITM). Seven proxies also accept RSA-512 keys in the leaf certificate. An attacker in possession of a valid certificate using a 512-bit RSA key for a website could recover the private key "at most in weeks" [9] and impersonate the website to the proxy. We could not test the behavior of AVG due to limitations explained in Section V-C.

Browser-trusted CAs are known to have stopped issuing RSA-512 certificates (some have even been sanctioned and distrusted for doing so, see e.g., [23]), and certificates using MD5 were not issued past 2008 [49]. Recently, Malhotra et al. [36] showed that attacks on the Network Time Protocol can trick a client system to revert its clock back in time by several years. Such attacks may revive expired certificates with weak RSA keys (easily broken), and weak hashing algorithms (i.e., re-enabling any certificate colliding with a previously-valid certificate, e.g., the colliding CA certificate forged in [60]).

3) Proxy-embedded trusted store: AVG, BitDefender, BullGuard IS, and Net Nanny solely rely on their own trusted stores. For Net Nanny, we managed to insert our root certificate in its encrypted database (see Section VI-B2). BullGuard IS

[9] An sqlite3 derivative: https://github.com/NuSkooler/ICUSQLite3.

[10] Note that, such databases can be encrypted using various ciphers, and the encryption key could be derived from the passphrase by an arbitrary number of iterations of SHA1 using PBKDF2; these parameters are unavailable to us. We failed to decipher the databases using the extracted passphrase with several common ciphers, and the number of iterations from 1 to half a million.

TABLE II. RESULTS OF THE CERTIFICATE VALIDATION PROCESS AGAINST 9 INVALID CERTIFICATES. FOR LEGENDS, SEE SECTION VI-C1; "N/A" MEANS NOT TESTED.

	Trusted store	Invalid certificate tests						
		Self-signed	Signature mismatch	Fake GeoTrust	Wrong CN	Unknown CA / Non-CA / v1 inter.	Revoked	Expired
Avast	OS	u-CA	Block	u-CA	Pass	u-CA / Block / Block	**Accept**	Mapped
AVG	Own	u-CA	N/A	N/A	Pass	u-CA / N/A / N/A	Unfiltered	Mapped
BitDefender	Own	u-CA	u-CA	u-CA	u-CA	u-CA	**Accept**	u-CA
BullGuard IS	Own	S-S	**Accept**	**Accept**	Pass	S-S	**Accept**	Mapped
CYBERsitter	None	**Accept**	**Accept**	**Accept**	**Change**	**Accept**	**Accept**	Mapped
Dr. Web	OS	u-CA	u-CA	u-CA	Pass	u-CA	**Accept**	u-CA
ESET	OS	Ask	Ask	Ask	Pass	Ask	**Accept**	Ask
G DATA (old)	None	**Accept**	**Accept**	**Accept**	**Change**	**Accept**	**Accept**	**Change**
G DATA (new)	None	Ask	Ask	Ask	Ask	Ask	Ask	Ask
Kaspersky	OS	Ask	Ask	Ask	Ask	Ask	Ask	Ask
KinderGate	None	**Accept**	**Accept**	**Accept**	Pass	**Accept**	**Accept**	**Change**
Net Nanny	Own	Ask	Ask	Ask	Ask	Ask	Ask	Ask
PC Pandora	None	**Accept**	**Accept**	**Accept**	Pass	**Accept**	**Accept**	**Change**

prevents modifications to its list of trusted CAs. If modified, it triggers an update to restore the original version. An option in its configuration allowed us to stop this protection. BitDefender adopts a similar mechanism, with no option to disable it; we bypassed this protection and changed the trusted store file by booting Windows in safe-mode (without BitDefender being started). Finally, more reverse-engineering is needed to make AVG accept our root certificate.

Except for AVG, we were able to retrieve all proxy-trusted CAs. BitDefender's trusted store contains 161 CA certificates, 41 with a 1024-bit key (most are now deprecated by browsers). As a comparison, Mozilla Firefox trusted store contains 180 certificates, including 13 RSA-1024 as of August 2015. Ten of BitDefender's trusted CA certificates have already expired as of August 2015; however, BitDefender does not accept certificates issued by an expired trusted root certificate. Most importantly, BitDefender's trusted store includes the DigiNotar certificate, distrusted by major browsers since August 2011, due to a security breach. It also includes the CNNIC certificate that was at the center of another breach in March 2015, subsequently distrusted by Firefox and Chrome.[11]

BullGuard IS trusted store was apparently generated in May 2009, from Mozilla's list of trusted CAs; as expected, this 6 year-old store has been outdated long ago. Among its 140 CAs, there is a CA with a 1000-bit key and 43 CAs with a 1024-bit key. Similar to BitDefender, BullGuard IS also includes the distrusted DigiNotar root certificate. It also fails at verifying the expiration dates of its root CAs during certificate validation, leaving the 13 expired root certificates in its store still active.

Net Nanny's trusted store contains 173 certificates; one CA with 512-bit key (named "Root Agency"), and 27 CAs with a 1024-bit key. Thus, Net Nanny is vulnerable to a generic MITM attacker, who can recover the private key for the 512-bit certificate (requires only trivial effort [9]). In addition, 16 CAs are expired, but Net Nanny effectively does not trust such root certificates when validating a site certificate.

D. TLS parameters

In this section, we provide the results of our analysis of TLS parameters; see Table III.

1) SSL/TLS versions: At the end of 2014, following the POODLE attack, major browsers dropped support for SSL 3.0 by default [53], [46], [59]. However, as of August 2015, we found half of the 12 proxies still support SSL 3.0.

Only Avast and Kaspersky support TLS 1.0, 1.1, 1.2, and map them appropriately; other proxies upgrade the SSL/TLS versions for the proxy-browser connection, and/or do not support recent versions. AVG, BitDefender and CYBERsitter upgrade all versions to TLS 1.2. G DATA also upgrades TLS 1.0, 1.1 and 1.2 to TLS 1.2. Net Nanny, which supports only SSL 3.0 and TLS 1.0 to connect to a server, communicates with TLS 1.2 with the browser. Similarly, BullGuard IS supports only TLS 1.0 but maps it to TLS 1.2 for browsers. Finally, Dr. Web, ESET, KinderGate and PC Pandora support only TLS 1.0, along with SSL 3.0 for the former two. The fictitious upgrade of TLS versions as done by a majority of these proxies mislead browsers to believe that the server provides stronger/weaker security than it actually does.

We test whether protocol downgrade attacks as seen against certain browser implementations are possible, and we found that no proxies in our test implement such a version downgrading. These proxies are thus not vulnerable to POODLE [41] via a downgrade attack. However, when connecting to servers that only support SSL 3.0 or lower, and offer CBC-mode ciphers, the practical padding oracle attack proposed in POODLE still applies to proxies with SSL 3.0. Six proxies accepted connections to such servers (disallowed by modern browsers) and presented the connections as TLS 1.0 or above to browsers.

We did not test whether the TLS proxies support SSL 2.0; note that, proxies that support SSL 2.0 (if any), may pose additional risks against servers that also support this version. For completeness, such testing may also be incorporated.

2) Certificate security parameters: All proxies, except Avast and PC Pandora, generate certificates with fixed RSA keys to communicate with browsers. Six use RSA-1024 and the remaining four use RSA-2048. While RSA-1024 still does not pose an immediate security risk, proxies may need to remove RSA-1024 to avoid warning/blocking by browsers (cf. [45]). Regarding the hashing algorithm used for the certificate signature, 7 proxies replace the original certificate's signing algorithm with SHA1, triggering security warnings in Chrome when the certificate expiration date is past December 31, 2015. BitDefender, ESET and Kaspersky use SHA256, effectively suppressing potential warnings for server certificates with SHA1 or MD5. Other proxies map hash algorithms properly.

3) Cipher suites: SSL 3.0 and TLS 1.0 support ciphers that are vulnerable to various attacks. For example, CBC-mode ciphers are vulnerable to the Lucky-13 and BEAST attacks;

[11] https://blog.mozilla.org/security/2015/03/23/revoking-trust-in-one-cnnic-intermediate-certificate/

TABLE III. Results for TLS parameters, proxy transparency and known attacks. Under "Protocol mapping" we list the TLS versions as observed by browsers when a TLS proxy connects to a server using TLS 1.2, 1.1, 1.0, SSL 3.0 ("—" means unsupported). For "Cipher suite problems", we use: "W" for weak (according to Qualys); "E" for export-grade ciphers; "A" for anonymous Diffie-Hellman. "×" represents vulnerability to the listed attacks; "*" indicates that the vulnerability to BEAST or FREAK could be due to the unpatched Schannel library used in our testing.

	Filtered ports	Protocol mapping				Certificate mapping			Vulnerabilities					
		TLS 1.2	TLS 1.1	TLS 1.0	SSL 3.0	Key size	Hash algorithm	EV cert.	Cipher suite problems	Insecure renego-tiation	BEAST	CRIME	FREAK	Logjam
Avast	Specific	1.2	1.1	1.0	—	Mapped	Mapped	Unfiltered						
AVG	Specific	1.2	**1.2**	**1.2**	**1.2**	2048	Mapped	DV	W					
BitDefender	Specific	1.2	**1.2**	**1.2**	**1.2**	2048	SHA256	DV	W		×			
BullGuard IS	Specific	**—**	**—**	**1.2**	—	1024	SHA1	DV	W		×			×
CYBERsitter	Specific	1.2	**1.2**	**1.2**	**1.2**	1024	SHA1	DV	W, E		×		×	
Dr. Web	All	**—**	**—**	1.0	**1.0**	1024	SHA1	DV	W		×*		×*	
ESET	Specific	**—**	**—**	1.0	**1.0**	2048	SHA256	DV	W		×*		×*	
G DATA	Specific	1.2	**1.2**	**1.2**	—	1024	SHA1	DV	A				×	×
Kaspersky	All	1.2	1.1	1.0	—	2048	SHA256	DV				×		
KinderGate	Specific	**—**	**—**	1.0	—	1024	SHA1	DV	W					
Net Nanny	All	**—**	**—**	**1.2**	**1.2**	1024	SHA1	DV	W		×		×	×
PC Pandora	All	**—**	**—**	1.0	—	Mapped	SHA1	DV	W	×	×			

and RC4 is known to have statistical biases [3]. To mitigate BEAST from the server-side, the preferred ciphers for SSL 3.0/TLS 1.0 were based on RC4. However, as modern browsers now mitigate this attack by using record splitting [61], servers continue to use CBC-mode ciphers in TLS 1.0 to avoid RC4 [54] (considering recent practical attacks against RC4 used in a TLS setting [68]).

We test TLS proxies for their supported cipher suites by using a browser that does not support any weak ciphers. When the Qualys test reports that weak ciphers are presented to the server, this indicates that the proxy negotiated its own cipher suite with problematic ciphers. Weak ciphers as ranked by the Qualys test include the ones relying on RC4, as presented by most proxies. Other used weak cipher suites include: export-grade ciphers with 40 bits of entropy (CYBERsitter); 56-bit DES (BullGuard IS and CYBERsitter); ciphers relying on anonymous Diffie-Hellman, which lacks authentication and may enable a generic MITM attack (G DATA). PC Pandora only supports three ciphers, two of which are based on RC4.

4) Known attacks: All proxies, except Avast, BitDefender (March 2015 version) and Kaspersky, are vulnerable to at least one of the following attacks: insecure renegotiation, BEAST, CRIME, FREAK, or Logjam.

BullGuard IS, CYBERsitter, Dr. Web, ESET, G DATA and Net Nanny are vulnerable to FREAK and/or Logjam against vulnerable servers. When the browser connects to a vulnerable server, an active MITM attacker could force the use of export-grade DH or RSA keys to access plaintext traffic. As of August 2015, 8.4% of servers from the Alexa Top 1 million domains are vulnerable to Logjam [1], and 8.5% to FREAK.[12] While Logjam and FREAK attacks are relatively recent (less than a year old at the time of our tests in August 2015), other attacks are known for several years. Kaspersky is vulnerable to CRIME; and PC Pandora to insecure renegotiation. In the latter case, an active MITM attacker could request server resources using the client's authentication cookies.

Although BEAST requires bypassing the Same-Origin Policy (SOP) and the support for Java applets, the main mitigation relies on Java's TLS stack implementation [54]. These mitigations are however canceled by five proxies that support TLS 1.0 at most (BullGuard IS, Dr. Web, ESET, Net Nanny and PC Pandora), since they do not implement proper mitigations with CBC (record splitting) or do not individually proxy each TLS record from the browser/Java client.

BullGuard IS, Dr. Web, ESET, Kaspersky, Net Nanny and PC Pandora may allow MITM attackers to decrypt partial traffic (typically authentication cookies, leading to session hijacking) because of their vulnerability to BEAST, CRIME, or insecure renegotiation.

VII. Practical attacks

In this section, we summarize how an attacker may exploit the reported vulnerabilities, and turn them into practical attacks against a target running Windows 7 SP1. For example, even if a CCA relies on a pre-generated root certificate, it may not become instantly vulnerable to a generic MITM attack. Other factors must also be considered, e.g., whether the certificate is imported in the OS/browser stores during installation, or later when the filtering option is enabled; whether the proxy is enabled after installation by default and in this case, if it accepts its own root certificate. We discuss such nuances when considering what attackers can realistically gain from the flaws we uncovered, and give a preliminary ranking of CCAs according to the level of effort required for launching practical attacks. We contacted the 12 affected companies; only four of them provided a detailed feedback, sometimes demonstrating a poor understanding of TLS security; see Appendix D.

An attacker who can launch a generic MITM attack can impersonate any server with very little or no effort to hosts that have any of the following four CCAs installed. (a) PC Pandora, as it imports a pre-generated root certificate in the Windows store during installation, and does not filter TLS traffic by default (i.e., allowing external site certificates signed by the PC Pandora private key to be directly validated by clients relying on the OS store, e.g., IE). It also remains vulnerable when filtering is enabled, as it accepts external certificates signed by its own root certificate. (b) KinderGate, for selected categories of websites, due to its lack of any certificate validation. (c) G DATA (for emails only), as the March version does not perform certificate validation, and both March/August versions support anonymous DH ciphers. (d) Net Nanny, as its March version uses a pre-generated certificate, and both March/August versions trust a root certificate with a factorable RSA-512 key (only one factorization is required to impersonate any server).

[12]https://freakattack.com/

The following three CCAs become vulnerable to full server impersonation when filtering is manually activated (disabled by default), or when the product's trial period is over. The attacker simply needs to wait for these attack opportunities, and requires no additional effort. (a) Kaspersky's March version, as it does not perform any validation after the product license is expired. Also, no automatic update of the product is possible (requires a valid license), thus leaving customers with the March version vulnerable until they manually upgrade or uninstall the product. (b) BullGuard IS, if the parental control feature is enabled, due to its lack of certificate signature validation. (c) CYBERsitter, when its TLS filtering option is enabled as it does not perform any certificate validation.

By exploiting the CRIME vulnerability, with limited effort (see e.g., [55]), attackers can retrieve authentication cookies under a generic MITM attack from hosts where Kaspersky is installed (both March/August versions). However, only the servers that still support TLS compression can be exploited. According to the SSL Pulse project [66], 4.4% of the TLS servers surveyed remain vulnerable, as of August 2015.

If attackers can launch the BEAST attack, they can retrieve authentication cookies from hosts with Dr. Web (out-of-the-box), ESET (when filtering is enabled) and BitDefender (both versions, for servers supporting at most TLS 1.0). As estimated [62], a PayPal cookie can be extracted using BEAST in about 10 minutes. According to SSL Pulse [66], 86.8% of TLS servers present CBC-mode ciphers in SSL 3.0/TLS 1.0, as of August 2015 (mostly due to mitigations being implemented in recent browsers, see e.g., [54]).

Attackers can exploit the FREAK attack against BitDefender's March version against servers that support TLS 1.1 or above (other FREAK-vulnerable CCAs can be exploited with simpler attacks). It will allow server impersonation for all websites served from a vulnerable web server. Note that 8.5% of Alexa's top 1 million domain names are reported to be vulnerable to FREAK, as of August 2015 [9].

If the attacker can execute unprivileged code on a target machine to retrieve private keys (not protected by the OS), she can further impersonate any server to seven CCAs (including BullGuard AV, BitDefender (August version) and ZoneAlarm). BullGuard IS and Kaspersky (March versions) could already be targeted by an opportunistic attack mentioned above, or the CRIME attack; however, a targeted attack requires no waiting and does not depend on server compatibility. BitDefender (March version), Kaspersky (August version) and Dr. Web can already be exploited for selected vulnerable websites, now it extends the attacker's ability to target any website. Finally, KinderGate also facilitates this attack, even after uninstallation (recall that KinderGate is already vulnerable to server impersonation under a generic MITM attack).

A more powerful attacker could further exploit RC4 weaknesses against systems with AVG installed (for selected websites only). More than 55% of servers surveyed by SSL Pulse in August 2015 present a cipher suite that includes RC4. The attack however is costly; it is reported by Vanhoef et al. [68] to require 75 hours to recover a single cookie.

For Avast, the only way to impersonate a server is to trick/compromise a CA to issue valid certificates for targeted websites. Even if the breach is later discovered and the certificates are revoked, Avast would continue to accept them.

VIII. Recommendations for safer TLS proxying

Encryption as provided by TLS is by design end-to-end, and insertion of any filtering MITM proxy is bound to interfere with TLS security guarantees. In this section, we discuss a few recommendations that may reduce negative interference of proxies/filtering. We also briefly discuss how browsers can help make proxying safer.

We first discuss the use of a special SSL key logging feature provided by recent browsers that would avoid the need for TLS proxies in CCAs, while allowing filtering to some extent. If proxies are still used (e.g., for clients without SSL key logging support), we then discuss how they may be designed to function safely. We believe following these guidelines may significantly improve CCAs in general, but we want to stress that more careful scrutiny is required to assess security, functionality and performance impacts. Note that, some TLS security features will be affected, no matter how the proxies are designed. For example, EV certificates cannot be served to browsers, if a proxy is used for filtering traffic from websites with EV certificates.

TLS key-logging. Recent Firefox and Chrome browsers support saving TLS parameters in a file to recreate a TLS session key that can be used to decrypt/analyze TLS traffic (e.g., via Wireshark); the key file is referenced by the SSLKEYLOGFILE environment variable [44]. TLS proxies can offload all TLS validation checks to browsers, by configuring the key file and using the session key to decrypt the TLS encrypted traffic originating from supporting browsers. Thus, proxies can passively intercept the traffic, and perform filtering as usual, without interfering with TLS security. This mechanism should be sufficient for antiviruses to protect browsers from active exploits, and parental control applications to block access to restricted content. We found no CCAs leveraging this functionality.

If TLS key logging is used, modification of the traffic may not be possible (e.g., censor swear words, remove ads). Also, browsers and other TLS applications (e.g., Microsoft IE, Safari, email clients) that currently do not support TLS key logging, cannot be filtered; note that, most CCAs filter traffic from selected applications only (see Table IV).

Private keys. Most CCAs attempt to manage their private keys independently (i.e., without relying on OS-protected storage), making the keys accessible to unprivileged code. Several keys are stored in plaintext, and others are protected by application-specific encryption/obfuscation techniques, which can be defeated with a one-time moderate effort. Instead, proxies can simply use the OS-provided API (CNG) to securely store private keys, which would then require an attacker to run admin-privileged code to access the keys. Of course, OS APIs should be used properly for effective protections (e.g., non-exportable key). Also, proxies must generate a separate root certificate for each installation, i.e., must never use a pre-generated certificate to avoid generic MITM attacks.

Certificate validation. To perform filtering, proxies must use dynamically generated server certificates for the proxy-browser TLS communication channel. Thus, proxies cannot transparently forward a server certificate to the browser. However, they must properly validate the received server certificates, with no less rigor than popular browsers, and relay certificate errors to browsers, as closely as possible. These are no easy tasks, but

must not be sidestepped by proxies, as they become the effective Internet-facing TLS engine for the filtered applications.

Validation: Proxies that perform validation checks (albeit incomplete), apparently rely on the validation mechanisms offered by their respective TLS library. Such mechanisms as provided by, e.g., OpenSSL, may require additional support to ensure the chain of trust, and revocation status, and to enforce supplementary policies.[13] The revocation status of certificates (via CRL or OCSP) should also be checked (e.g., through the OpenSSL `ocsp` interface).

Errors: Communicating non-critical validation errors such as expired certificate or wrong CN should be done in a way that users still have a choice to accept or reject them, similar to common browsers. Other invalid scenarios, e.g., non-CA and X.509v1 intermediate, could also be replicated; however, simply refusing such certificates might also be acceptable (reflecting how browsers deal with such error cases).

Transparency. For the browser-proxy connection, proxies should not use a fixed-size key or a fixed hashing algorithm, which we observed for most products. When certificate attributes are not properly mapped, browsers may remain unaware of the true TLS security level of an intended server. Achieving transparency of certificate attributes includes at least the replication of the same signature hashing algorithm and key type/size. Regarding the TLS version and other parameters such as the cipher suite, a transparent TLS handshake is possible that satisfies constraints from both the browser and server. Below, we outline a simple protocol to achieve this goal; see also Fig. 2.

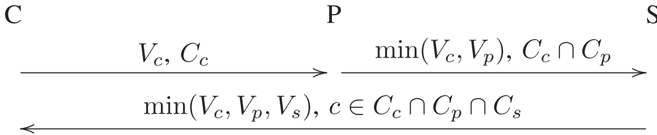

Fig. 2. Optimal handshake for TLS ClientHello and ServerHello when proxying a connection

In this three-party TLS handshake, the client (C) sends a ClientHello message with its supported TLS version (V_c) and cipher suite (C_c). The proxy (P) intercepts the message and attempts a connection with the remote server (S) using the best version that both the client and the proxy support, i.e., $\min(V_c, V_p)$, along with a cipher suite that is compatible with both the client and proxy ($C_c \cap C_p$). Finally, the server naturally chooses a TLS version and a cipher (c) that would transparently satisfy both the proxy and the client, i.e., $\min(V_c, V_p, V_s)$ and $c \in C_c \cap C_p \cap C_s$ respectively (V_s is the best version supported by the server and C_s is the server's cipher suite). The proxy simply relays the ServerHello message to the client, and continues the two handshakes (client- and server-end) separately.

The proxy achieves complete transparency, if its supported cipher suite is a superset of the client's ($C_p \supseteq C_c$), and if it supports at least a TLS version as high as the client ($V_p \geq V_c$). Such a handshake requires the proxy to be at par with the latest TLS standards. This requirement is also necessary to help deter newly discovered attacks (e.g., Heartbleed,[14] FREAK).

Recommendations for browser manufacturers. As TLS filtering obviously breaks end-to-end security, we recommend a few additional active roles for browsers, specifically, to reduce harm from broken proxies. For example, browsers can warn users when a root certificate is inserted to a browser-specific trusted store (e.g., the Firefox store), or when filtering is active (e.g., via a warning page, once in each browsing session); connections via proxies may also be contingent upon user confirmation. Such warnings may be undesirable for parental-control applications, which may be mitigated by having the warning feature as an option, turned on by default. At least, browsers should make active filtering apparent to users through security indicators. Note that browsers can easily detect the presence of proxies, e.g., from the received proxy-signed certificate, and recent browsers already accommodate several UI indicators, to show varying levels of trust in a given TLS connection.[15] Some users may ignore such indicators, but others may indeed be benefited (cf. [2]). Recently, Ruoti et al. [56] surveyed user attitudes toward traffic inspection, and reported that users are generally concerned about TLS proxies (in organizations, public places, or operated by the government); 90.7% of participants expected to be notified when such proxying occurs.

As the most used interface to web, browser manufacturers in the recent years have taken a more pro-active role in improving online security than simply faithfully implementing the TLS specifications, e.g., deploying optional/experimental extensions to TLS, such as HSTS and key pinning; blocking malware and phishing sites; and restricting misbehaving CAs, such as CNNIC [4] and TURKTRUST [48]. We thus expect browser manufacturers to force companies behind the most offending CCAs to fix obvious vulnerabilities, by blocking connections when a known, broken proxy is involved.

IX. Related work

Most testing suites related to our framework are presented in Section IV. Here we briefly report additional studies on TLS interception, proxying, and TLS security in general.

Dell SecureWorks Counter Threat Unit [16] propose a framework for testing dedicated, network-based TLS interception appliances as used in enterprise environment; several security flaws were also reported. CERT [19] lists a few common vulnerabilities in TLS proxies, and identifies possibly affected products (mostly for enterprises). In the past, such devices used to receive certificate signing authority from an existing client-trusted CA to avoid user configuration; however, many OS/browser vendors disallow this practice, and have removed/sanctioned the issuing CA when discovered, e.g., Trustwave [63], TURKTRUST [48], ANSSI [47] and CNNIC [4]. Such enterprise proxies require users/administrators to independently install the proxy's root certificate into their clients. Our work is focused on client-end interception proxies, which poses additional challenges, and are installed and used by everyday users. Also, Dell's framework is mostly oriented towards certificate validation, while we extend the focus to TLS versions and various recent attacks.

Frankencert [12] generates artificial certificates that are composed of a combination of existing extensions and con-

[13] https://www.openssl.org/docs/apps/ocsp.html, /docs/apps/verify.html
[14] http://heartbleed.com/

[15] See e.g., Chrome: https://support.google.com/chrome/answer/95617; and Firefox: https://support.mozilla.org/en-US/kb/how-do-i-tell-if-my-connection-is-secure.

straints, randomly chosen from a large corpus of input certificates. The generated certificates are then tested against TLS clients. Errors are uncovered through differential testing between at least two implementations. Frankencert has been tested mainly on open-source TLS libraries (not much testing on browsers), and uncovered several high-impact validation flaws. The authors use a script to instrument browsers and TLS libraries to generate a web request and log the status of the reply (i.e., to check certificate rejection errors). We provide a simple mechanism to make Frankencert compatible with client-end TLS proxies; however, we do not use/modify Frankencert as obvious validation errors are already apparent from simple tests.

In a preliminary work, Böck [11] analyzes three antiviruses, and reports that they are vulnerable to CRIME and FREAK attacks, and support only old SSL/TLS versions. Böck also tracks commercial products that leverage the Netfilter SDK[16] to intercept HTTPS traffic using pre-generated certificates. Our work is more comprehensive in terms of the number of tested products, and tests we perform in our framework.

Huang et al. [30] study TLS traffic filtering by investigating Facebook's server certificate as seen from browsers. They found that 0.2% of the 3 million TLS connections they measured were tampered with interception tools, mostly antiviruses and enterprise CCAs, but also parental control tools and malware. O'Neill et al. [51] leverage a Google AdWords campaign to study connections to their own server and several popular websites. They found that 0.41% of 15 million connections were proxied, by similar types of intercepting tools.

Various proposals introduce extensions to TLS and new encryption schemes that enable transparent inspection of encrypted traffic, see e.g., [58], [50]. Liang et al. [35] show the architectural difficulties faced by CDNs to deploy HTTPS, as they are automatically placed in a man-in-the-middle position.

Meyer and Schwenk [37] survey theoretical and practical cryptographic attacks against SSL/TLS, along with problems with the PKI infrastructure. They gather lessons learned from these attacks, e.g., the need for reliable cryptographic primitives and awareness for side-channel attack origins. In parallel, Clark and van Oorschot [13] survey issues related to SSL/TLS from a cryptographic point of view in the context of HTTPS, as well as general issues related to current PKI and trust model proposals. Recent proposals, e.g., key pinning and HSTS variants, OCSP stapling and short-lived certificates, have also been evaluated against known issues. Authors note a shift from cryptographic attacks against TLS to attacks on the trust model, where valid certificates can be issued by attackers.

HTTP Strict Transport Security (HSTS [31]) is a simple mechanism to protect against SSL stripping attacks. Kranch and Bonneau [34] studied how HSTS and key pinning are deployed in practice, and found that even such simple proposals to enhance the HTTPS security are challenging to implement. We note that key pinning is overridden by Chrome 47.0 when the server certificate is signed by an imported root certificate.

Huang et al. [29] study the deployment of forward secrecy (FS) compatible ciphers from the server perspective, and found that despite their wide-scale adoption, weak parameters (weak keys) are still often negotiated. We did not test whether TLS proxies interfere with such FS-ciphers.

[16]http://netfiltersdk.com/

X. Conclusion

We propose a framework for the evaluation of client-end TLS proxies, by addressing limitations of regular TLS test suites, and adding more tests specifically relevant to such proxies. We use the framework to comprehensively analyze 14 antiviruses and parental control applications, specifically their TLS proxies. While these applications may require TLS interception capabilities for their functionality, they must avoid introducing new weaknesses into the already fragile browser/SSL ecosystem. However, we found that not a single TLS proxy implementation is secure with respect to all of our tests, sometimes leading to trivial server impersonation under an active man-in-the-middle attack, as soon as the product is installed on a system. Our analysis calls the purpose of such proxies into question, especially in the case of antiviruses, which are tasked to enhance host security. Indeed, these products in general, appear to significantly undermine the benefits of recent security fixes and improvements as deployed in the browser/SSL ecosystem. We suggest preliminary guidelines for safer implementations of TLS proxies based on our findings. However, due to the foreseeable implementation complexities of our proposed guidelines, we suggest the adoption of interfaces that would let client-end TLS proxies monitor encrypted traffic originating from browsers in a more secure way, e.g., using the SSL key log file feature. Our work is intended to highlight weaknesses in current TLS proxies, and to motivate better proposals for safe filtering. Finally, our findings also call into question the so-called security best-practice of using antiviruses on client systems, as commonly advised by IT professionals, and even required by some online banking websites.

Acknowledgments

For comments and suggestions, we are grateful to anonymous CCS2015 and NDSS2016 reviewers, Paul Van Oorschot, Jeremy Clark, Tao Wan, our shepherd Joseph Bonneau, and the members of Concordia's Madiba Security Research Group. The first author is supported in part by a Vanier Canada Graduate Scholarship (CGS). The second author is supported in part by an NSERC Discovery Grant and an OPC Contributions Program (Office of the Privacy Commissioner of Canada).

References

[1] D. Adrian, K. Bhargavan, Z. Durumeric, P. Gaudry, M. Green, J. A. Halderman, N. Heninger, D. Springall, E. Thomé, L. Valenta, B. VanderSloot, E. Wustrow, S. Zanella-Béguelink, and P. Zimmermann, "Imperfect forward secrecy: How Diffie-Hellman fails in practice," in *CCS'15*, 2015.

[2] D. Akhawe and A. P. Felt, "Alice in warningland: A large-scale field study of browser security warning effectiveness," in *USENIX Security Symposium*, 2013.

[3] N. J. AlFardan, D. J. Bernstein, K. G. Paterson, B. Poettering, and J. C. Schuldt, "On the security of RC4 in TLS," in *USENIX Security Symposium*, 2013.

[4] ArsTechnica.com, "Google Chrome will banish Chinese certificate authority for breach of trust," news article (Apr. 1, 2015). http://arstechnica.com/security/2015/04/google-chrome-will-banish-chinese-certificate-authority-for-breach-of-trust/.

[5] ——, "Lenovo PCs ship with man-in-the-middle adware that breaks HTTPS connections," news article (Feb. 19, 2015).

[6] AV-comparatives.org, "Independent tests of anti-virus software - summary reports," http://www.av-comparatives.org/summary-reports/.

[7] ——, "Parental control reviews," http://www.av-comparatives.org/parental-control/.

[8] M. Benham, "IE SSL vulnerability," Bugtraq mailing list (Aug. 5, 2002). http://seclists.org/bugtraq/2002/Aug/111.

[9] 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 S&P*, 2015.

[10] H. Böck, "Check for bad certs from Komodia/Superfish," https://superfish.tlsfun.de/.

[11] ——, "How Kaspersky makes you vulnerable to the FREAK attack and other ways antivirus software lowers your HTTPS security," https://blog.hboeck.de/archives/869-How-Kaspersky-makes-you-vulnerable-to-the-FREAK-attack-and-other-ways-Antivirus-software-lowers-your-HTTPS-security.html.

[12] C. Brubaker, S. Jana, B. Ray, S. Khurshid, and V. Shmatikov, "Using frankencerts for automated adversarial testing of certificate validation in SSL/TLS implementations," in *IEEE S&P*, 2014.

[13] J. Clark and P. C. van Oorschot, "SSL and HTTPS: Revisiting past challenges and evaluating certificate trust model enhancements," in *IEEE S&P*, 2013.

[14] Comodo.com, "Comodo SSL affiliate the recent RA compromise," blog article (Mar. 23, 2011). https://blog.comodo.com/other/the-recent-ra-compromise/.

[15] ComputerWeekly.com, "PrivDog SSL compromise potentially worse than Superfish," news article (Apr. 24, 2015).

[16] Dell.com, "SSL/TLS interception proxies and transitive trust," http://secureworks.com/cyber-threat-intelligence/threats/transitive-trust/.

[17] B. Delpy, "mimikatz," http://blog.gentilkiwi.com/.

[18] DigiCert.com, "Apache SNI browser support," https://www.digicert.com/ssl-support/apache-secure-multiple-sites-sni.htm.

[19] W. Dormann, "The risks of SSL inspection," online article (Mar. 13, 2015). https://www.cert.org/blogs/certcc/post.cfm?EntryID=221.

[20] T. Duong and J. Rizzo, "Here come the $\oplus$ ninjas," technical report (May 2011). http://www.hpcc.ecs.soton.ac.uk/~dan/talks/bullrun/Beast.pdf.

[21] DuoSecurity.com, "Dude, you got Dell'd," technical report (Nov. 24, 2015). https://duosecurity.com/static/pdf/Dude,_You_Got_Dell_d.pdf.

[22] Z. Durumeric, E. Wustrow, and J. A. Halderman, "ZMap: Fast internet-wide scanning and its security applications." in *USENIX Security Symposium*, 2013.

[23] D. Fisher, "Malaysian CA Digicert revokes certs with weak keys, Mozilla moves to revoke trust," news article (Nov. 3, 2011). https://threatpost.com/malaysian-ca-digicert-revokes-certs-weak-keys-mozilla-moves-revoke-trust-110311/75847.

[24] Google, "Certificate transparency," http://certificate-transparency.org.

[25] ——, "Gradually sunsetting SHA-1," blog article (Sept. 5, 2014). http://googleonlinesecurity.blogspot.ca/2014/09/gradually-sunsetting-sha-1.html.

[26] R. D. Graham, "Extracting the SuperFish certificate," http://blog.erratasec.com/2015/02/extracting-superfish-certificate.html.

[27] ——, "Heartleech," https://github.com/robertdavidgraham/heartleech.

[28] J. Hodges, "howsmyssl," https://github.com/jmhodges/howsmyssl.

[29] L. S. Huang, S. Adhikarla, D. Boneh, and C. Jackson, "An experimental study of TLS forward secrecy deployments," *Internet Computing, IEEE*, vol. 18, no. 6, pp. 43–51, 2014.

[30] L. S. Huang, A. Rice, E. Ellingsen, and C. Jackson, "Analyzing forged SSL certificates in the wild," in *IEEE S&P*, 2014.

[31] IETF, "Internet-Draft: HTTP strict transport security (HSTS)," 2012, RFC 6797 (Standards Track).

[32] A. Junestam, C. Clark, and J. Copenhaver, "Jailbreak 4.0," https://github.com/iSECPartners/jailbreak.

[33] G. Kopf and P. Kehrer, "CVE-2011-0228 – iOS certificate chain validation issue in handling of X.509 certificates."

[34] M. Kranch and J. Bonneau, "Upgrading HTTPS in mid-air: An empirical study of strict transport security and key pinning," in *NDSS'15*.

[35] J. Liang, J. Jiang, H. Duan, K. Li, T. Wan, and J. Wu, "When HTTPS meets CDN: A case of authentication in delegated service," in *USENIX Security Symposium*, 2014.

[36] A. Malhotra, I. E. Cohen, E. Brakke, and S. Goldberg, "Attacking the Network Time Protocol," in *NDSS'16*, 2016.

[37] C. Meyer and J. Schwenk, "SoK: Lessons learned from SSL/TLS attacks," in *Information Security Applications (WISA'13)*, 2013.

[38] Microsoft, "CA certificates tools and settings," https://technet.microsoft.com/en-us/library/cc783813%28v=ws.10%29.aspx.

[39] ——, "Key storage and retrieval," https://msdn.microsoft.com/en-us/library/windows/desktop/bb204778%28v=vs.85%29.aspx.

[40] ——, "System store locations," https://msdn.microsoft.com/en-us/library/windows/desktop/aa388136%28v=vs.85%29.aspx.

[41] B. Moeller, T. Duong, and K. Kotowicz, "This POODLE bites: Exploiting the SSL 3.0 fallback," technical report (Sept. 2014). https://www.openssl.org/~bodo/ssl-poodle.pdf.

[42] Mozilla, "Dates for phasing out MD5-based signatures and 1024-bit moduli," wiki article (Oct. 3, 2013). https://wiki.mozilla.org/CA:MD5and1024.

[43] ——, "Mozilla CA certificate policy," https://www.mozilla.org/en-US/about/governance/policies/security-group/certs/policy/.

[44] ——, "NSS key log format," https://developer.mozilla.org/en-US/docs/Mozilla/Projects/NSS/Key_Log_Format.

[45] ——, "Phasing out certificates with 1024-bit RSA keys," blog article (Sept. 8, 2014). https://blog.mozilla.org/security/2014/09/08/phasing-out-certificates-with-1024-bit-rsa-keys/.

[46] ——, "The POODLE attack and the end of SSL 3.0," blog article (Oct. 14, 2014). https://blog.mozilla.org/security/2014/10/14/the-poodle-attack-and-the-end-of-ssl-3-0/.

[47] ——, "Revoking trust in one ANSSI certificate," blog article (Dec. 13, 2013). https://blog.mozilla.org/security/2013/12/09/revoking-trust-in-one-anssi-certificate/.

[48] ——, "Revoking trust in two TurkTrust certificates," blog article (Jan. 3, 2013). https://blog.mozilla.org/security/2013/01/03/revoking-trust-in-two-turktrust-certficates/.

[49] P. Mutton, "Governments and banks still using weak MD5-signed SSL certificates," news article (Aug. 31, 2012). http://news.netcraft.com/archives/2012/08/31/governments-and-banks-still-using-weak-md5-signed-ssl-certificates.html.

[50] D. Naylor, K. Schomp, M. Varvello, I. Leontiadis, J. Blackburn, D. R. López, K. Papagiannaki, P. Rodriguez Rodriguez, and P. Steenkiste, "Multi-Context TLS (mcTLS): Enabling secure in-network functionality in TLS," in *SIGCOMM'15*, 2015.

[51] M. O'Neill, S. Ruoti, K. Seamons, and D. Zappala, "TLS proxies: Friend or foe?" http://arxiv.org/abs/1407.7146v3.

[52] Qualys, Inc., "SSL/TLS capabilities of your browser," https://ssllabs.com/ssltest/viewMyClient.html.

[53] M. Qureshi, "April 2015 security updates for Internet Explorer," blog article (Apr. 14, 2015).

[54] I. Ristić, "Is BEAST still a threat?" blog article (Sept. 10, 2013). https://community.qualys.com/blogs/securitylabs/2013/09/10/is-beast-still-a-threat.

[55] J. Rizzo and T. Duong, "The crime attack," in *Ekoparty*, 2012, http://netifera.com/research/crime/CRIME_ekoparty2012.pdf.

[56] S. Ruoti, M. O'Neil, D. Zappala, and K. Seamons, "At least tell me: User attitudes toward the inspection of encrypted traffic," https://isrl.byu.edu/pubs/ruoti2016at.pdf.

[57] M. Russinovich, "Inside Windows 7 User Account Control," 2009, magazine article. https://technet.microsoft.com/en-us/magazine/2009.07.uac.aspx?rss_fdn=TNTopNewInfo.

[58] J. Sherry, C. Lan, R. A. Popa, and S. Ratnasamy, "BlindBox: Deep packet inspection over encrypted traffic," in *SIGCOMM'15*, 2015.

[59] Softpedia.com, "Chrome 39 disables SSLv3 fallback," news article (Nov. 19, 2014).

[60] A. Sotirov, M. Stevens, J. Appelbaum, A. Lenstra, D. Molnar, D. A. Osvik, and B. de Weger, "MD5 considered harmful today," blog article (Dec. 30, 2008). https://www.win.tue.nl/hashclash/rogue-ca/.

[61] X. Su, "(CVE-2011-3389) Rizzo/Duong chosen plaintext attack (BEAST) on SSL/TLS 1.0 (facilitated by websockets -76)," https://bugzilla.mozilla.org/show_bug.cgi?id=665814#c59.

[62] TheRegister.co.uk, "Hackers break SSL encryption used by millions of sites," news article (Sept. 19, 2011). http://www.theregister.co.uk/2011/09/19/beast_exploits_paypal_ssl/.

[63] ——, "Revoking trust in two TurkTrust certificates," news article (Feb. 14, 2012). http://www.theregister.co.uk/2012/02/14/trustwave_analysis/.

[64] TLS-O-Matic.com, "Self testing for web and application developers," https://www.tls-o-matic.com/.

[65] TopTenReviews.com, "Parental software review," http://parental-software-review.toptenreviews.com/.

[66] Trustworthy Internet Movement, "SSL Pulse," survey (retrieved on Aug. 3, 2015). https://www.trustworthyinternet.org/ssl-pulse/.

[67] F. Valsorda, "Superfish, Komodia, PrivDog vulnerability test," https://filippo.io/Badfish/.

[68] M. Vanhoef and F. Piessens, "All your biases belong to us: Breaking RC4 in WPA-TKIP and TLS," in *USENIX Security Symposium*, 2015.

TABLE IV. SECURITY ASPECTS RELATED TO ROOT CERTIFICATES INSERTION/REMOVAL, AND FILTERING

	Certificate generation time	Filtering enrollment	Reject own root certificate	Insertion in Firefox trusted store	Removal during uninstallation	Filtered clients
Avast	Installation	Mandatory		✓	✓	Internet Explorer, Chrome, Firefox
AVG	Installation	Mandatory	✓[1]		✓	Internet Explorer, Chrome
BitDefender	Installation	Mandatory		✓	✓	Internet Explorer, Chrome, Firefox
BullGuard AV	Installation	Unsupported	—	✓		—
BullGuard IS	Installation	Opt-in	✓	✓		All
CYBERsitter	Pre-generated[2,3]	Opt-in		✓		All
Dr. Web	Installation	Mandatory				All
ESET	Installation[3]	Opt-in		✓		All
G DATA	Installation	Mandatory			✓	All
Kaspersky	Installation	Mandatory		✓		Internet Explorer, Chrome, Firefox
KinderGate	Installation	Mandatory				All
Net Nanny	Installation	Mandatory		✓	✓	Internet Explorer, Chrome, Firefox
PC Pandora	Pre-generated	Opt-in			✓	Internet Explorer
ZoneAlarm	Installation	Unsupported	—			—

[1] The product does not filter connections with a proxy-signed certificate, leaving clients to accept the certificate
[2] A pre-generated public key is wrapped in a new certificate during its creation
[3] A root certificate is installed when the relevant option is activated (and removed when deactivated for ESET)

APPENDIX

A. *Trusted root CA stores*

System CA store. All versions of Windows starting from Windows 2000 [38], provide a Trusted Root Certification Authorities certificate store that comes preloaded with a list of trusted CAs, meeting the requirements of the Microsoft Root Certificate Program.[17] Updates to this list are generally provided by Microsoft, but applications and users can add additional certificates (only via specific Windows APIs or the Windows Certificate Manager). We refer to this store as the OS trusted (CA) store, which can either be user-dependent, service-dependent or machine-wide. The machine-wide trusted store is located in Windows registry as (key, value) pairs [40]: a key (*Certificates*) hosting each trusted certificate as a subkey, labeled with the certificate's SHA1 fingerprint; and a value (*Blob*) hosting the certificate in the ASN.1 DER format. CCAs import their root certificates in the machine-wide store, making those certificates trusted by the OS and all applications relying on the OS trusted store. Importing a root certificate into the machine-wide store requires admin privileges, in which case Windows does not warn users about the security implications of such a certificate. Importing a root certificate to the current-user's trusted store by a userland application however triggers a detailed warning, and requires explicit user acceptance. As CCAs obtain admin privileges during installation (e.g., via a UAC prompt), the insertion of a root certificate into the OS trusted store remains transparent to the user.

Third-party CA stores. TLS applications may choose to use their own CA store, instead of relying on the OS-provided store (possibly due to not fully trusting the validation process as used by Microsoft to accept a root certificate). For example, Firefox uses an independent root CA list, populated according to the Mozilla CA Certificate Policy [43]. In addition to the OS store, several CCAs also insert their root certificates into the application stores to filter traffic to/from those applications. CCAs may check for such applications during installation, and automatically insert their root certificates into selected third-party stores (transparently to users), or simply instruct users to manually add root certificates to application stores.

Table IV summarizes which CCA (from the list of tested products in Table V) imports its root certificate in Firefox trusted store, along with various details discussed in Section VI-A.

[17] https://technet.microsoft.com/en-ca/library/cc751157.aspx

B. *OS-provided APIs for key storage*

The legacy Microsoft CryptoAPI (CAPI) and the new Cryptography API: Next Generation (CNG) provide specialized functions to store, retrieve, and use cryptographic keys [39]. Cryptographic Service Providers (CSP) such as the Strong Cryptographic Provider in the previous CAPI, and the CNG Key Storage Provider (KSP) offer such features. For TLS filtering, CCAs must store their private keys (corresponding to their root certificates) in the host system to sign site certificates for browsers on-the-fly. If a CCA uses CSP/KSP to securely store its private key, Windows encrypts the private key using a master key only available to the OS, and stores the ciphertext in %ProgramData%\Microsoft\Crypto\RSA\MachineKeys in the case of machine-wide RSA private keys. For CCAs using CSP/KSP, we check whether a key is marked as exportable (by the CCA). Machine-wide keys are exportable only with admin privileges. If a key is marked non-exportable, it is not supposed to be exported even with admin privileges. However, tools requiring admin/system privileges are available to bypass this restriction, e.g., Jailbreak [32] and Mimikatz [17] as we tested on Windows 7 SP1. Non-exportable keys can be used by the CAPI or CNG to directly encrypt or decrypt data without letting the application access the key. Such a method should be preferred by CCAs; however our results show otherwise (see Section VI). In this paper, we consider that exporting OS-protected private keys requires admin privileges. Note that, an unprivileged application running under an admin account, can open the Windows Certificate Manager (run with admin privileges), and then instrument the UI to access an exportable private key; such an attempt will not trigger the Windows UAC prompt under default UAC settings (under Windows 7, 8.1 and 10 as we tested), which allow auto-elevating whitelisted Microsoft tools [57].

C. *Test certificates with a broken chain of trust*

1) Self-signed: A simple self-signed certificate. If accepted, trivial generic MITM attacks are possible.
2) Signature mismatch: The signature of a valid certificate is altered. If accepted, the proxy lacks signature verification, and may allow simple certificate forgery.
3) Fake GeoTrust CA: A certificate signed by an untrusted root certificate that has the same subject name as the GeoTrust root CA (any OS/browser trusted CA can be used). We also include this fake CA certificate in the

certificate chain. The leaf certificate does not specify an Authority Key Identifier (AKI), limiting the identification of the issuer certificate to only its subject name. The goal is to check if the proxy refers to the correct root certificate.

4) Wrong CN: Incorrect Common Name (CN) not matching the domain where it is served from. If accepted, a valid certificate for *any* website could be used to impersonate *any* server.
5) Unknown CA: A certificate signed by an untrusted root certificate (e.g., generated by us).
6) Non-CA intermediate: A valid leaf certificate is used as an intermediate CA to sign a new certificate. If accepted, a valid certificate for *any* website could be used to issue valid certificates for *any* other websites (cf. early versions of IE [8] and iPhone [33]).
7) X.509v1 intermediate: An X.509 version 1 certificate acting as an intermediate CA certificate. X.509v1 does not support setting a *basicConstraints* parameter to limit a certificate to be a leaf. If accepted, *any* valid v1 certificate could be used to issue *any* other certificates.
8) Revoked: We rely on https://revoked.grc.com to test the revocation support. This website delivers a revoked certificate with the necessary extensions to refer to the signing CA's CRL list and OCSP server (both would report the certificate as revoked). Revocation is particularly useful in cases where legitimate certificates are issued after a security breach at a CA, e.g., Comodo [14].
9) Expired: A certificate with a past "valid-before" date.

D. Company responses

The companies behind the products that we tested are listed in Table V. We contacted all affected companies except Avast (as its lack of revocation checking is not serious enough). Among the 12 emails we sent, we received an acknowledgment from seven companies (beyond a simple automatic reply), and received a detailed reply in four cases. Among these four replies, two antivirus companies were already aware of the bugs we reported and had fixed them in more recent releases of their software. One reply from a parental control software company highlighted several discrepancies and misconceptions. For example, our tests on the latest version of the product on Windows 7 SP1 with patches for Schannel against BEAST and FREAK reveal that it supports at most TLS 1.0 when connecting to remote websites. However, the company states that "*In fact, Net Nanny supports up to TLS v1.2.*", and further adds that the "**real* server connection is established with the highest settings we can use without being rejected.*" Also, while the FREAK attack is an implementation flaw in some TLS libraries that allows an attacker to force both parties to agree on export-grade ciphers, the company states that "*FREAK and logjam are again, due to having to support old browsers/servers.*" The last parental control software company simply downplayed the risks as their software does not filter sensitive websites by default (but can be configured to do so). They wrote: "*That's why our users are not affected by any vulnerability or MITM-attack.*" Finally, the companies behind the most offending products did not reply after four months, even after a reminder.

TABLE V. LIST OF PRODUCTS TESTED. **HIGHLIGHTED** ENTRIES ARE PRODUCTS THAT MAY INSTALL A ROOT CERTIFICATE AND PROXY TLS CONNECTIONS; WE ANALYZED ALL SUCH PRODUCTS.

Company	Product	Version
	Antiviruses	
Agnitum	Outpost Security Suite Pro	9.1
AhnLab	V3 Internet Security	8.0
Avast	**Internet Security**	**2015 10.2.2218**
		10.3.2225
AVG	**Internet Security**	**2015.0.?**
		2015.0.6122
Baidu	Antivirus	2015 5.0.3
BitDefender	**Antivirus Plus**	**2015 v8**
BullGuard	**Antivirus**	**15.0.297**
	Internet Security	**15.1.302**
		15.1.307.2
Checkpoint	**ZoneAlarm Security Suite**	**2015 13.4.261**
Comodo	Antivirus Advanced	8.1
	Internet Security	8.1
CMC	Internet Security	2012
Dr. Web	**Security Space**	**10**
Emsisoft	Anti-Malware	9.0
eScan	Internet Security Suite	14.0
ESET	**Smart Security**	**8.0.312.0**
		8.0.319.0
F-Secure	SAFE	2.15 build 364
G DATA	**Antivirus**	**2015 25.0.0.2**
		25.1.0.3
K7 Computing	K7 Internet Security	14.2.0.249
	K7 Total Security Pro	14.2.0.249
Kaspersky	**Antivirus**	**15.0.2.361**
		16.0.0.614
Kingsoft	Antivirus	2010
McAfee	Internet Security	12.8
Norman	Security Suite	11
Output	Total Security	1.1.4304.0
Panda Security	Antivirus Pro	2015
	Internet Security	2015
Qihoo	360 Internet Security	5.0.0.5104
	360 Total Security	6.0.0.1140
Quick Heal	Internet Security	16.00 (9.0.0.20)
Sophos	Endpoint Security	10.3
TGSoft	VirIT	Lite 7.8.51.0
Total Defense	Internet Security Suite	9.0.0.141
TrendMicro	Internet Security	8.0
TrustPort	Total Security	2014 14.0.5.5273
	Internet Security	2015 15.0.3.5432
VIPRE	Internet Security	2015 8.2.1.16
Webroot	SecureAnywhere	8.0.7.33
	Parental control applications	
Awareness Tech	WebWatcher	8.2.30.1147
BlueCoat	K9 Web Protection	4.4.276
ContentWatch	**Net Nanny**	**7.2.4.2**
		7.2.6.0
Cybits Ag	JuSProg	6.1.0.106
Fortinet	FortiClient	5.2
Entensys	**KinderGate Parental Control**	**3.1.10058.0.1**
KinderServer AG	KinderServer	1.1
LavaSoft	Ad-Aware Total Security	11
McAfee	SafeEyes	6.2.119.1
Norton	Family	3.2.1
Pandora Corp	**PC Pandora**	**7.0.22**
Profil	Parental Filter	2
Salfeld	Child Control	2014 14.644
Solid Oak Software	**CYBERsitter**	**11**
SpyTech	SpyAgent	8
TuEagles	AntiPorn	2.15
Verify	Parental Control	1.15
Witigo	Parental Filter	?