The Age of Encryption

GIAC GCIA Gold Certification

Author: Wes Whitteker, wes_whitt@yahoo.com

Advisor: Richard Carbone

Accepted: November 2\textsuperscript{nd}, 2016

Abstract

Over the last few years, there has been an increasing movement toward encrypting Internet communication. Though this movement increases the confidentiality of transmitted information, it also severely limits the ability of security tools to analyze Internet traffic for malicious content. This paper investigates the growth of encrypted Internet traffic (i.e. HTTPS) and its impact on Cybersecurity. This paper also proposes an open source solution for decrypting and inspecting Internet traffic accommodating IPv4 and v6 for both home and small-to-medium sized business (SMB) use.
1. Introduction

Since the beginning of the Internet, encryption has been in use to protect sensitive information such as financial data, personal data, etc. As the Internet has evolved, the use of encryption has been steadily increasing in parallel through the use of both the Secure Sockets Layer (SSL) and Transport Layer Security (TLS) protocols (Netcraft, n.d.). Over the last few years, this increase in SSL and TLS Internet communication has been very significant due in large part to disclosures and disputes that have created public concern for privacy. These privacy concerns have created additional momentum around the growth of SSL and TLS communication on the Internet (IETF, 2014 & Letsencrypt.org, n.d.).

This increased momentum for encrypting Internet communication is creating a significant dilemma for cybersecurity professionals. Since networking is the glue that pulls all computing technologies together, it serves as a key environment for cybersecurity professionals to identify malicious activity. However, the burst of growth in encrypted communication is creating “blind spots.” Often times, cybersecurity professionals need deep visibility into the traffic traversing their networks in order to determine if communication is legitimate or illegitimate. Unfortunately, this analytical capability is severely hampered when communication is encrypted (Mayfield, 2015).

There have been several examples of areas where malicious actors and malware are leveraging the growth in encryption to their benefit. In a 2015 report, it was shown that malware command and control servers using SSL and TLS communication increased by 200 times, and malware samples using encrypted communication increased by 58 times. When considering these data points, it is clear that cybersecurity professionals are facing a significant challenge around encrypted communication analysis. Cybersecurity professionals can address this challenge by decrypting and inspecting SSL and TLS communication (A10 Networks, 2015, Davis, 2016, & Segura, 2015).

Wes Whitteker, wes_whitt@yahoo.com
2. Overview of the Internet

To understand the current state of SSL and TLS communication, it is important to briefly look at the history of the Internet and its use of both SSL and TLS. The following subsections provide an overview of the creation of the World Wide Web (WWW) and the usage of SSL and TLS with the Hypertext Transfer Protocol (HTTP), commonly referred to as “HTTPS.” For the remainder of the paper, SSL and TLS will be collectively referred to as “TLS” unless referred to specifically by version.

2.1. World Wide Web and HTTP Background

The inception of the Internet can be traced back to the Cold War, which created a sense of urgency within the United States to increase their focus on science and technology developments. This initiative resulted in the establishment of several key organizations and the publishing of several academic research papers that laid the foundation for packet switched networks. This foundation led to the standardization of the Transmission Control Protocol/Internet Protocol (TCP/IP) suite; the first Domain Name System (DNS); the Internet Assigned Numbers Authority (IANA); and the Internet Engineering Task Force (IETF) (History.com, n.d., Pew Research Center, n.d., World Wide Web Foundation, n.d., & World Wide Web Consortium, n.d.).

In 1991, CERN introduced the World Wide Web (WWW) to the masses (World Wide Web Foundation, n.d., & World Wide Web Consortium, n.d.). The component of the WWW that is used to transfer data is the Hyper Text Transfer Protocol (HTTP). The IETF began oversight of the HTTP in 1996 (IETF, 1996). The latest IETF recommended version of HTTP is 1.1, which became an IETF draft standard in June of 1999 (IETF, 1999 & W3C, 2004). The core of the HTTP 1.1 protocol is now covered as a proposed standard under two IETF Request for Comments (RFCs): RFC7230 & RFC7231 (IETF HTTP WG, n.d.). The latest approach to HTTP is HTTP/2, which is an extension of the Google SPDY project. HTTP/2 is now an IETF proposed standard (IETF, 2015). The intent of HTTP/2 is to improve upon the overall performance of HTTP 1.1 (e.g. less TCP connections). However, HTTP/2 is being advertised by the IETF as an alternative to HTTP 1.1 instead of obsoleting it (IETF HTTP WG, n.d.a.).
HTTP/2 does not require encryption for use. However, there are no browsers currently being developed that support HTTP/2 without the use of encryption. HTTP/2 does seek to improve TLS based security by defining a profile that includes a specific version of TLS, a cipher suite blacklist, and extensions used (IETF HTTP WG, n.d.a.). Specifically, HTTP/2 requires that TLS 1.2 or greater is used, and recommends the use of RFC 7525, Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS). In addition to the TLS version minimum being set to 1.2, HTTP/2 also uses a blacklist for cipher suites that can be found here: https://http2.github.io/http2-spec/#BadCipherSuites (IETF HTTP WG, n.d.b.).

Further, under HTTP/2, TLS must leverage Server Name Indication (SNI), which requires that a client specify the domain name with which it intends to communicate. SNI is explained under RFC 6066, Transport Layer Security (TLS) Extensions: Extension Definitions (IETF HTTP WG, n.d.b). If HTTP/2 uses TLS 1.3 or higher the use of only SNI is required. However, if TLS 1.2 is used, SNI is still required but there are additional restrictions. The restrictions under TLS 1.2 include disabling compression; disabling renegotiation (except prior to connection preface information); support for ephemeral key exchange sizes of at least 2048 bits for cipher suites that use ephemeral finite field Diffie-Hellman (DHE) and 224 bits for cipher suites that use ephemeral elliptic curve Diffie-Hellman (ECDHE); and clients need to accept DHE sizes of up to 4096 bits (IETF HTTP WG, n.d.b).

2.2. SSL and TLS Background

Though HTTP has done an outstanding job in formatting and transmitting information for the WWW, it only offers basic authentication capabilities and does not offer a mechanism for encryption (IETF, 1999b). The primary response to this was the development of SSL. In 1994, Netscape, supported by MasterCard, Bank of America, MDI & Silicon Graphics, introduced SSL (Hwang, 2012, Ristic, 2014, & Wireshark.org, n.d.). Today, the SSL and TLS (successor protocol to SSL) protocols provide the mechanisms necessary for confidentiality, authentication, integrity, and non-repudiation around much of the secure communications on the modern WWW (CIO.gov, n.d. & Evsslcertificate.com, n.d.).

Wes Whittleker, wes_whitt@yahoo.com
The first public release of SSL was version 2 with version 3 being released in 1996 (IETF, 2011). It was at this time that the IETF began to officially acknowledge SSL by the establishment of the TLS working group (IETF TLS WG, n.d.). The TLS working group used SSL 3.0 as the foundation for developing the TLS 1.0 protocol. In 1999, the IETF published TLS 1.0 under RFC 2246 (IETF, 1999a). Since the initial release of TLS there have been 2 additional versions released, TLS 1.1 (RFC 4346) and TLS 1.2 (RFC 5246) (IETF, 2006 & IETF, 2008). Currently, there is a draft of TLS 1.3 (IETF, 2016). Due to security issues in previous versions of both SSL and TLS, the IETF encourages the use of TLS 1.2 through RFC 7568, which also officially deprecates SSL 3.0 (IETF, 2015a). Further, the IETF offers a specific RFC for using TLS in a secure manner (IETF, 2015b).

In addition to the IETF recommending the use of TLS version 1.2, the US National Institute of Standards and Technology (NIST) recommends the use of TLS version 1.2 as well (Chokhani, McKay, Polk, 2014). The recommendation for moving to TLS 1.2 is due, in large part, to security issues (wolfSSL.com, 2010). This can be seen when comparing the changes of TLS 1.1 and TLS 1.2. The following tables offer the major improvements in TLS versions 1.1 and 1.2 as well as the security updates to TLS 1.2:

<table>
<thead>
<tr>
<th>Table 1: Major updates to TLS 1.1 and TLS 1.2 (IETF, 2006 &amp; IETF, 2008)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TLS 1.1 Major Updates</strong></td>
</tr>
<tr>
<td>The implicit Initialization Vector (IV) is replaced with an</td>
</tr>
<tr>
<td>explicit IV to protect against CBC attacks.</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Handling of padding errors is changed to use the bad_record_</td>
</tr>
<tr>
<td>mac alert rather than the decryption_failed alert to protect</td>
</tr>
<tr>
<td>against CBC attacks.</td>
</tr>
<tr>
<td>IANA registries are defined for protocol parameters.</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Premature closes no longer cause a session to be nonresumable.</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Additional informational notes were added for various</td>
</tr>
<tr>
<td>new attacks on TLS.</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

Wes Whitteker, wes_whitt@yahoo.com
Cleaned up description of Bleichenbacher/Klima attack defenses.

Alerts MUST now be sent in many cases.

After a certificate_request, if no certificates are available, clients now MUST send an empty certificate list.

TLS_RSA_WITH_AES_128_CBC_SHA is now the mandatory to implement cipher suite.

Added HMAC-SHA256 cipher suites.

Removed IDEA and DES cipher suites. They are now deprecated and will be documented in a separate document.

Support for the SSLv2 backward-compatible hello is now a MAY, not a SHOULD, with sending it a SHOULD NOT. Support will probably become a SHOULD NOT in the future.

Added limited "fall-through" to the presentation language to allow multiple case arms to have the same encoding.

Added an Implementation Pitfalls sections

The usual clarifications and editorial work.

Table 2: Security updates to TLS 1.2 (IETF, 2008)

<table>
<thead>
<tr>
<th>RFC Number</th>
<th>RFC Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>RFC 5746</td>
<td>TLS Renegotiation Indication Extension</td>
</tr>
<tr>
<td>RFC 5878</td>
<td>TLS Authorization Extensions</td>
</tr>
<tr>
<td>RFC 6176</td>
<td>Prohibiting SSL Version 2.0</td>
</tr>
<tr>
<td>RFC 7465</td>
<td>Prohibiting RC4 Cipher Suites</td>
</tr>
<tr>
<td>RFC 7507</td>
<td>TLS Fallback Signaling Cipher Suite Value (SCSV) for Preventing Protocol Downgrade Attacks</td>
</tr>
<tr>
<td>RFC 7568</td>
<td>Deprecating Secure Sockets Layer Version 3.0</td>
</tr>
<tr>
<td>RFC 7627</td>
<td>TLS Session Hash and Extended Master Secret Extension</td>
</tr>
<tr>
<td>RFC 7685</td>
<td>A TLS ClientHello Padding Extension</td>
</tr>
<tr>
<td>RFC 7905</td>
<td>ChaCha20-Poly1305 Cipher Suites for TLS</td>
</tr>
<tr>
<td>RFC 7919</td>
<td>Negotiated Finite Field Diffie-Hellman Ephemeral Parameters for TLS</td>
</tr>
</tbody>
</table>

As can be clearly seen when looking at the improvements in the list of major updates to TLS 1.1 and 1.2, TLS version 1.2 has had a significant amount of work completed around its security functionality. In general, TLS 1.1 added support for TLS extensions and TLS 1.2 added support for authenticated encryption and removed hard-coded security features (Ristic, 2015).

2.3. HTTP and TLS

The parent IETF RFC that ties TLS to HTTP is 2818, though TLS can be used with several other popular Internet protocols (IETF, 2000 & Wireshark.org, n.d.). RFC 2818 has been updated by both RFCs 7230 and 5785. In addition, RFC 2817 provides

Wes Whitteker, wes_whitt@yahoo.com
guidance on upgrading an HTTP connection to an HTTPS connection, and RFC 2817 has been updated by both RFCs 7230 and 7231.

3. Technicalities of HTTPS Communication on the Internet

It is extremely important for cybersecurity professionals looking for a possible HTTPS decryption solution to understand the protocol at a detailed level. As such, the following section will cover the key technical aspects of the TLS protocol. The major areas of discussion will be the Public Key Infrastructure (PKI) and the TLS protocol.

3.1. Public Key Infrastructure (PKI)

Much of the HTTPS communication performed today relies on what is known as the Public Key Infrastructure (PKI). Today’s PKI is based on an architecture standard known as X.509. At the core of the Internet’s PKI are certificate authorities, which are trusted to issue certificates for use in several key TLS encryption functions. There are four key entities in PKI known as the Subscriber, Registration Authority (RA), Certification Authority (CA), and the Relying Party. The subscriber is an entity that desires to support secure communication (i.e. the website owner). An RA performs many of the management activities needed for certification issuance such as verifying an identity. A CA is the key to the Internet’s PKI because it is trusted with issuing and revoking certificates. The relying party is normally associated with an end user’s web browser. The web browser performs validation of certificates by leveraging a local root trust store. If a website’s public certificate is received from a CA the browser has a root trust store for, the browser will trust the website certificate, otherwise the user will receive a browser message that the certificate is unknown or untrusted (Risitc, 2015).

3.1.1. Certificates

The overall reason the PKI has been built is to share certificates that are used to establish trust between a server and a web browser. Certificates are essentially a digital document with several specific fields and associated values depending on the version of the certificate. Certificates are formatted based on a standard known as the Abstract
Syntax Notation One (ASN.1) and often communicated in Privacy-Enhanced Mail (PEM) encoding (Ristic, 2015). Table 4 provides the key fields seen in a certificate.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Field Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>Provides the specific certificate version of either 1, 2, or 3.</td>
</tr>
<tr>
<td>Serial Number</td>
<td>Uniquely identifies the CA of the certificate.</td>
</tr>
<tr>
<td>Signature Algorithm</td>
<td>Provides the signature algorithm used for the certificate signature.</td>
</tr>
<tr>
<td>Issuer</td>
<td>Contains a distinguished name (DN) of the certificate issuer.</td>
</tr>
<tr>
<td>Validity</td>
<td>Provides the length of time that the certificate is valid via both a start and end date.</td>
</tr>
<tr>
<td>Subject</td>
<td>The DN for the entity associated with the public key. This field has been deprecated for Subject Alternative Name Extension Field.</td>
</tr>
<tr>
<td>Public Key</td>
<td>The public key.</td>
</tr>
<tr>
<td>Extensions</td>
<td>Extensions were added in certificate version 3 to ensure future flexibility. There are now several extensions available for use including: Subject Alternative Name, Name Constraints, Basic Constraints, Key Usage, Extended Key Usage, Certificate Policies, CRL Distribution Points, Authority Information Access, Subject Key Identifier, and Authority Key Identifier.</td>
</tr>
</tbody>
</table>

Certificates are issued using a specific process associated with the type of certificate requested. There are three processes used for certificate issuance: Domain Validation (DV), Organizational Validation (OV), and Extended Validation (EV). EV is the most rigorous and was developed out of security issues with the other two processes. Once the appropriate validation process is completed, the requesting party receives the certificate. In order for a relying entity to validate certificates, they maintain a list of “trusted” root CA certificates. In general, most applications (e.g. browsers) rely on the underlying operating system to manage certificate trust store. However, there are cases where applications will not rely on the OS trust store (e.g. Mozilla & Chrome) (Ristic, 2015).


The two methods of certificate revocation are Certificate Revocation Lists (CRL) and the Online Certificate Status Protocol (OCSP). The CRL provides a list of revoked certificates...
certificates that are still unexpired – these are large lists. The OCSP addresses the CRL issue of large lists by looking up a certificate’s status in real-time. To address some of the security concerns around the TLS and the current PKI architecture, there have been several initiatives to improve these weaknesses. Two of the most significant developments have been HTTP Strict Transport Security (HSTS) and Certificate Pinning (Ristic, 2015).

HSTS is a standard covered under IETF RFC 6797 and is focused on ensuring the confidentiality and authenticity of communication between a browser and a server. This capability mitigated the SSL Stripping attack presented by Moxie Marlinspike at Blackhat in 2009 (Marlinspike, n.d.). The two key functions of HSTS are it forces the browser to use HTTPS communication instead of HTTP and it does not allow users to click through browser certificate warnings. In terms of decrypting TLS traffic, this could create some issues (e.g. browser security warnings) depending on the specific architecture. The simplest way to mitigate possible issues is to leverage a Man-In-The-Middle (MITM) decryption solution and install a trusted certificate used by all internal system browser-trusted stores (superconfigure.wordpress.com, 2013).

The key feature with pinning is that it allows domain owners to specify specific CAs that are authorized to issue certificates for their domains. This prevents an illegitimate entity from creating unauthorized certificates for a domain of their choosing. There are 2 basic types of certificate pinning: Hard Certificate Pinning and CA Pinning. Hard certificate pinning requires the exact match of the server certificate presented to the client or the communication is prevented. Hard certificate pinning is rarely used because many enterprises block it and there could be complications if the hard coded certificate needs changed. CA Pinning, on the other hand, trusts a limited set of CAs. As such, this type of pinning can be successfully decrypted using a root certificate that is manually added to system browser-trusted stores. This allows MITM capabilities for encrypted traffic inspection (Bluecoat, 2016 & Ristic, 2015).

### 3.2. The TLS Protocol

From a high level, TLS is broken into two major protocols known as the record protocol and the handshake protocol. The record protocol manages the transportation, encryption and integrity, compression, and extensibility feature of the TLS protocol. The Wes Whitteker, wes_whitt@yahoo.com
handshake protocol negotiates the connection settings and authentication between the client and server. The handshake protocol consists of three “subprotocols” referred to as the change cipher spec protocol, application data protocol, and the alert protocol. The handshake protocol can take three forms: full handshake including server authentication, handshake with client/server authentication, and abbreviated handshake (resuming an earlier session) (Ristic, 2015).

The main difference between a full handshake with server only authentication and a full handshake with both server and client authentication (i.e. mutual authentication) is in the latter method the server requests a certificate from the client as part of its handshake messages. In response to the servers’ additional request, the client provides its certificate and verifies it is the owner of the certificate. The handshake that happens during a session resumption leverages the Session IDs contained in the ClientHello and ServerHello messages. The benefit of this handshake method is that it saves time in establishing a new session by cutting the amount of handshake message in half. The most common handshake is the full handshake with server authentication. Figure 1 provides a depiction of the TLS handshake and Table 4 provides its description.

Wes Whitteker, wes_whitt@yahoo.com
Figure 1: TLS Handshake (Ristic, 2015)

Table 4: Handshake Description (Ristic, 2015)

<table>
<thead>
<tr>
<th>Step Number</th>
<th>Step Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step 1</td>
<td>The client sends a ClientHello message, which includes the following fields: Protocol Version, Random, Session ID, Cipher Suites, Compression, and Extensions. The protocol version provides the latest TLS protocol version supported by the client. The random field provides randomness during the handshake, which assists in authentication and integrity functions. The session ID is used when session is resumed but blank on an initial connection. The Cipher Suites field provides the list of client supported suites. Compression provides a listing of client supported compression methods. Extensions provide a listing of additional functionality used within the current communication session. The latest list of extensions can be found here: <a href="http://www.iana.org/assignments/tls-extensiontype-values/tls-extensiontype-values.xhtml">http://www.iana.org/assignments/tls-extensiontype-values/tls-extensiontype-values.xhtml</a>.</td>
</tr>
<tr>
<td>Step 2</td>
<td>The server sends a ServerHello message, which is like the ClientHello message in Step 1 except it limits the options per field to one value.</td>
</tr>
</tbody>
</table>

Wes Whitteker, wes_whitt@yahoo.com
### 4. Decryption Solution

#### 4.1. Solution Selection

There are several open source projects underway that offer the ability to design a decryption solution for specific environments. Some of these projects include the following: Mitmproxy, Sssl sniff, SSLSplit, SSLDump, and Fiddler. For the solution proposed by this paper, the following requirements were used as selection criteria: open source software (OSS); actively supported; scalable for home and SMB use; transparently decrypt and inspect egress HTTPS traffic (i.e. Forward Proxy); support for IPv4 & v6; support TLS 1.2; support for the most common TLS Ciphers, and easily configurable. Taking these requirements into consideration, the most sensible solution was to set up a pfSense firewall running Squid Proxy as a Man-In-The-Middle (MITM). With the MITM configuration the Internet Content Adaptation Protocol (ICAP), discussed in RFC 3507 can be used to analyze decrypted HTTPS content. The specific implementation of ICAP used in Squid Proxy is C-ICAP (C-ICAP project, n.d.). Essentially, pfSense intercepts the HTTPS traffic on port 443 and routes it through Squid Proxy, which performs the decryption and analysis of the HTTP content.

Wes Whitteker, wes_whitt@yahoo.com
4.2. Conceptual Network Architecture

A diagram of the test network architecture is shown in Figure 2. As can be seen, all Internet network traffic is routed directly through the pfSense system. This solution requires the generation of a certificate used by all internal system browser-trusted stores, which is a common approach for a decryption solution. The major drawback to this solution is the inability to dynamically intercept HTTPS traffic on ports other than 443 (some commercial solutions do offer this ability).

Figure 2: Test Network Architecture

4.3. Solution Setup and Configuration

PfSense is a very popular FreeBSD-based firewall and router that offers functional extensibility through additional packages (pfSense, n.d.). The latest version of pfSense can be downloaded from: https://www.pfsense.org/download/. One of the packages that comes integrated with pfSense is Squid Proxy. Squid Proxy is a very popular software proxy that has the ability to decrypt TLS sessions (Squid-cache.org, Wes Whittleker, wes_whitt@yahoo.com
n.d.). One key reason Squid Proxy incorporates decryption capability is to analyze HTTPS traffic for malicious content.

When setting up this solution you have to consider the resources needed for both the pfSense firewall and the Squid Proxy. The hardware specifications for a pfSense firewall installation are located here: [https://www.pfsense.org/hardware/](https://www.pfsense.org/hardware/). There is no “master” specification listing for Squid Proxy because every environment varies significantly. However, Squid Proxy is very CPU and Memory intensive. To this end, there are a few webpages that provide an overview for understanding what HW requirements might be: [http://wiki.squid-cache.org/BestOsForSquid](http://wiki.squid-cache.org/BestOsForSquid); [http://wiki.squid-cache.org/SquidFaq/SquidMemory](http://wiki.squid-cache.org/SquidFaq/SquidMemory); and [http://wiki.squid-cache.org/SquidFaq/SquidProfiling](http://wiki.squid-cache.org/SquidFaq/SquidProfiling).

The following configuration and setup information assumes that the basic pfSense installation has already been completed. With the pfSense system completely configured for basic functionality, it is now time to setup Squid Proxy for decrypting TLS communication. The first step is to add the Squid Proxy package to pfSense, which is simple with the pfSense package manager. Start by navigating to System > Package Manager.

Wes Whitteker, wes_whitt@yahoo.com
Select the “+ Install” button beside the squid package.

Select the “Confirm” button.
The installation will complete.

Now that Squid Proxy has been installed, it needs to be configured. Navigate to Services > Squid Proxy Server.

Once within the Services > Squid Proxy Server section, the following Squid General Settings should be selected: Enable Squid Proxy; Keep Settings/Data; and LAN &

Wes Whittleker, wes_whitt@yahoo.com
Loopback Proxy Interfaces. Nothing else needs to be completed within the General Settings section.

In the next section, Transparent Proxy Settings, enable Transparent HTTP Proxy and ensure LAN is the selected interface. This will allow for the seamless proxying of traffic on port 443 (i.e. TLS). If Transparent HTTP Proxy is not enabled, each browser would need to be configured to use the proxy and its assigned port.

Wes Whitteker, wes_whitt@yahoo.com
Once the general and transparent proxy settings are set, they need to be saved. Note: You may need to save the Local Cache settings before you can save the settings for the General and Transparent Proxy settings. At this point, you should test that your internal system/s web browsers can still access the Internet.

With the basic proxy functionality operational, it is now time to configure the Squid Proxy Man-In-The-Middle functionality. The first step is to setup pfSense as a Certificate Authority (CA). Navigate to System > Cert. Manager.

Wes Whitteker, wes_whitt@yahoo.com
Ensure CA is selected and click the “+ Add” button. Fill in the desired descriptive name and select the desired method. For the test environment, Create an internal Certificate Authority is selected.

For the actual CA settings, the defaults are used and the location information is filled in. Once finished click Save.

Wes Whitteker, wes_whitt@yahoo.com
Now the CA is setup with its root certificate, which will need to be exported and installed in all browser Trusted Root Certificate stores. Select the export CA button to export the .crt certificate.

The exported .crt certificate needs to be installed in the browser Trusted Root Certification store to prevent browser warnings since MITM proxying is being used. This can be completed manually or automated for larger environments. The manual steps for Windows 7 are described. On the system where you want the certificate installed, download the certificate to the system. Double click on the certificate and select Install Certificate.

Wes Whitteker, wes_whitt@yahoo.com
Click Next.

Wes Whitteker, wes_whitt@yahoo.com
Select “Trusted Root Certification Authorities”.

Click Finish.
Select Yes.

Go into Internet Explorer and verify the certificate in the Trusted Root Certification Authorities store.

Wes Whitteker, wes_whitt@yahoo.com
With the root certificate installed on the proper internal systems, Squid Proxy MITM functionality can be enabled. Navigate to Services > Squid Proxy Server and scroll down to “SSL Man In the Middle Filtering”. Once there, enable HTTPS/SSL Interception; select the LAN SSL Intercept Interface; choose the proper CA (SEC503 for the test environment). Click the Save button at the bottom of the page.

Using Internet Explorer, navigate to an encrypted webpage and look at the certificate being used. As you can see below, the certificate is actually for our internal pfSense CA.

Wes Whitteker, wes_whitt@yahoo.com
With the TLS decryption enabled and working, the AV scanning can be enabled. Navigate to Services > Squid Proxy Server > Antivirus. Select the Enable checkbox. For the test environment client forward options, the option was selected to not send any data to ClamAV. Make sure to change the ClamAV Database update period to the desired setting – every hour was selected for the test environment. At this point, it is worth noting that the ClamAV default signatures do not have the best reputation for detections. As such, it is recommended to augment the default signatures with the signatures from sansecurity.com (http://sanesecurity.com/usage/signatures). The freshclam.conf file will need updated with the locations for the desired additional signatures. This can be completed by adding the URL locations to the following freshclam.conf directive: DatabaseCustomURL. In order to update these signatures, you will have to load the advanced configuration.

Wes Whitteker, wes_whitt@yahoo.com
The AV definition update status can be reviewed by navigating to Services > Squid Proxy Server > Real Time, and scroll down to the “freshclam Table” section.

With the AV functionality setup, it is now time to test that it is working as planned. We will use the standard EICAR test signature. These are offered via both HTTP and HTTPS connections (www.eicar.org/85-0-Download.html). Below you can see the results when the HTTPS MITM functionality is enabled and the results when the HTTPS MITM functionality is disabled, respectively. You can see that the EICAR file is not blocked when MITM functionality is disabled. It is worth noting that the EICAR HTTPS site certificate is self-signed so you may get prompted with browser warnings.

Wes Whitteker, wes_whitt@yahoo.com
Additionally, there is a setting in Squid Proxy under Services > Squid Proxy Server > SSL Man In the Middle Filtering > Remote Cert Checks where you will need to select “Accept remote server certificate with errors”. If this option is not checked within Squid Proxy, you will receive a self-signed certificate error in Squid Proxy.

Another way to assess the decryption functionality between the Squid Proxy Service and the C-ICAP extension is to leverage tcpdump on the pfSense system. Below tcpdump is

Wes Whitteker, wes_whitt@yahoo.com
being used to find the string “123456,” which is found in the password field of a TLS encrypted login page:

Command used: tcpdump -i lo0 -Xnvv dst port 1344 | grep <insert text>

This command dumps the traffic on port 1344 of the local interface (used by the C-ICAP extension). It prints the output in HEX & ASCII, does not do name resolution and produces verbose output while grepping for a specified string: “http://www.tcpdump.org/tcpdump_man.html.”

At this point, the solution is both decrypting TLS traffic and inspecting it for potential viruses. There are obviously several other features and configuration options available in both pfSense and Squid Proxy. However, this is one possible solution that could be leveraged to address encrypted communication concerns.

5. Conclusion

As the paper has shown, the existing trend of TLS usage is increasing. This trend is being reinforced because of significant public concern for privacy. The inevitable result will be a very large growth in TLS communication. This “Age of Encryption” will have an obvious impact on cybersecurity. To this end, the ability for security analysts to do their jobs is challenged because they lack the visibility to comprehensively detect malicious activity on their networks. The bottom line is the path to more encryption has opened a major avenue for malicious activity.

To combat this, organizations can turn to decrypting both inbound and outbound Internet communications. There are certainly risks and rewards when organizations are deciding on a decryption solution. The overarching risk is “seeing” sensitive data. The Wes Whittleker, wes_whitt@yahoo.com
sensitive data could be corporate data, personal info, or any other data that is of a sensitive nature. The overarching benefit is that decryption provides insight into the embedded encrypted data of the communication. This content could be a file, URL, commands, or any other information that could be associated with malicious activity.

Due to the sensitive nature of an organization’s decision to decrypt Internet communication, it is important that businesses involve legal departments in the decryption strategy decision making process. Further, organizations need to comprehensively understand the data on their networks. In many cases the decryption decision making process will result in a balancing act between policy and security. This balancing act will require that organizations develop an appropriate strategy on traffic inspection within the confines of what is legally authorized (Boss, 2016, Butler, 2013, Casey, 2013, & Dormann, 2015).

Once organizations have a comprehensive decryption policy in place and a firm understanding of their network traffic, they can move forward with deciding on a solution. There are several open source decryption alternatives available, which allows some flexibility when determining what is needed for a particular situation. If an open source solution is not a workable option, there are also several commercial options available. Throughout the decision-making process, it is important that the technical details of the TLS communication process are considered to ensure the final solution is successful.

Wes Whitteker, wes_whitt@yahoo.com
References


Wes Whittaker, wes_whitt@yahoo.com


Wes Whitteker, wes_whitt@yahoo.com


Wes Whitteker, wes_whitt@yahoo.com


Wes Whitteker, wes_whitt@yahoo.com


Wes Whitteker, wes_whitt@yahoo.com


Wes Whitteker, wes_whitt@yahoo.com


Wes Whitteker, wes_whitt@yahoo.com

Wes Whitteker, wes_whitt@yahoo.com
# Upcoming Training

<table>
<thead>
<tr>
<th>Event</th>
<th>Location</th>
<th>Dates</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>SANS London October 2020</td>
<td>London, United Kingdom</td>
<td>Oct 12, 2020 - Oct 17, 2020</td>
<td>CyberCon</td>
</tr>
<tr>
<td>SANS October Singapore 2020 - Live Online</td>
<td>Singapore, Singapore</td>
<td>Oct 12, 2020 - Oct 24, 2020</td>
<td>CyberCon</td>
</tr>
<tr>
<td>SANS Dallas Fall 2020</td>
<td>Dallas, TX</td>
<td>Oct 19, 2020 - Oct 24, 2020</td>
<td>CyberCon</td>
</tr>
<tr>
<td>SANS London November 2020</td>
<td>, United Kingdom</td>
<td>Nov 02, 2020 - Nov 07, 2020</td>
<td>CyberCon</td>
</tr>
<tr>
<td>SANS San Diego Fall 2020</td>
<td>San Diego, CA</td>
<td>Nov 16, 2020 - Nov 21, 2020</td>
<td>CyberCon</td>
</tr>
<tr>
<td>SANS Frankfurt November 2020</td>
<td>, Germany</td>
<td>Nov 30, 2020 - Dec 05, 2020</td>
<td>CyberCon</td>
</tr>
<tr>
<td>SANS Cyber Defense Initiative 2020</td>
<td>Washington, DC</td>
<td>Dec 14, 2020 - Dec 19, 2020</td>
<td>CyberCon</td>
</tr>
<tr>
<td>SANS OnDemand</td>
<td>Online</td>
<td>Anytime</td>
<td>Self Paced</td>
</tr>
<tr>
<td>SANS SelfStudy</td>
<td>Books &amp; MP3s Only</td>
<td>Anytime</td>
<td>Self Paced</td>
</tr>
</tbody>
</table>