Snort and SSL/TLS Inspection

GIAC (GCIA) Gold Certification

Author: Yousef Bakhdlaghi, bakhdlaghi@gmail.com
Advisor: Sally Vandeven

Accepted: April 14th, 2017

Abstract

An intrusion detection system (IDS) can analyze and alert on what it can see, but if the traffic is tunneled into an encrypted connection, the IDS cannot perform its analysis on that traffic. The difficulty of looking into the packet payload makes the encrypted traffic one of the challenging issues to IDS. In Snort, the encrypted traffic inspector is available optionally and can only inspect connections' handshakes with no further inspection of the payload after the connection has established. However, encrypted traffic can be entirely decrypted using the private key (decryption key), but there are some issues associated with SSL/TLS key exchanges that could increase the difficulty of decrypting traffic provided the private key.

This work discusses SSL/TLS protocols, and the issues of key exchange methods in addition to providing solutions for inspecting SSL/TLS traffic with the demonstration of two methods to inspect SSL/TLS traffic.

Table of Contents

1. Introduction

An intrusion detection system (IDS) can analyze and alert on what it can see, but if the traffic is tunneled through an encrypted connection, an IDS can only perform limited inspection based on packet headers. The difficulty of looking into the packet payload makes the encrypted traffic one of the challenging issues for an IDS. From Snort point's of view, the encrypted traffic inspector, SSL Dynamic Pre-processor (SSLPP), is available optionally in Snort and can only inspect connections' handshakes, but once the encrypted connection has established, Snort will not perform any inspection on data for that connection (Snort FAQ, 2016). However, encrypted traffic can be entirely decrypted using the private key (decryption key) to decrypt and inspect the payloads (Juniper, 2010). But there are some issues associated with SSL/TLS key exchanges that could increase the difficulty of decrypting traffic provided the private key. The rest of this section briefly discusses Snort and its components as well as SSL/TLS key exchange and the possible ways to inspect encrypted connections.

1.1. Snort IDS

Snort is a free open-source IDS solution that offers intrusion detection and prevention capabilities (IDS/IPS) for firms as a cost-effective solution. It has the ability to perform real-time traffic analysis that attempts to detect malicious activity, in addition to content analysis and packet logging.

1.1.1. Snort Components

Snort consists of several components (Kannan, 2011) (Caswell, Beale, & Baker, 2007): packet sniffer, pre-processor, detection engine, and logging/alerting module as shown in *Figure 1.*

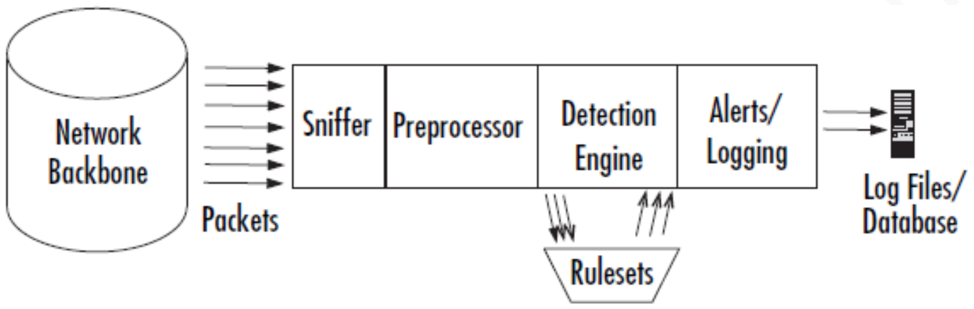


Figure 1: *Snort Architecture (Caswell, Beale, & Baker, 2007)*

Packet sniffer/decoder: Allows Snort to eavesdrop on the network interface and decodes all captured network traffic to be sent to the pre-processor.

Pre-processor: It operates on the decoded packet and performs a variety of transformations simplifying the data to be easier for Snort to process. It has several plug-ins that have the option to be enabled or disabled. For example, *frag3* pre-processor that defragments packets prior to sending the data on to the detection engine. This allows the detection engine to analyze the full packet stream for malicious behavior that might otherwise go unnoticed if passed through in smaller fragments.

Detection engine: It is the most important component of Snort that utilizes the rules/signatures to determine whether or not a packet matches a rule/signature. The rule is divided into two parts. The first part is the rule header that has the details about the action that Snort needs to execute for matching the incoming packets, while the second part is the options field that has additional information for rule matching to determine which portion of the packet should be used to fire an alert.

Logging and alerting: After detecting a malicious packet or activity, Snort triggers an alert. Depending on the alert configuration, Snort can send the alert using a variety of options such as: log file, database, and e-mail.

1.1.2. SSL Dynamic Pre-processor (SSLPP)

This pre-processor enables Snort to inspect SSL/TLS handshakes of each connection with no further data inspection, which is by default disabled. It inspects the unencrypted portion of the connection (headers) for faulty encrypted traffic to ensure two things: "*the last client-side handshake packet was not crafted to evade Snort, and that the traffic is legitimately encrypted*" (Snort FAQ, n.d.).

1.2. SSL/TLS

Secure Sockets Layer (SSL) and Transport Layer Security (TLS) are cryptographic protocols that were developed to secure communications on computer networks. Both are very similar protocols; however, TLS is the successor of SSL, which has been deprecated by IETF (RFC6176, 2011) (RFC7568, 2015). Currently, TLS is most commonly used to secure connections, however, many people still use the old name, SSL to refer to TLS (Ristić, 2015). *Figure 2* shows how the TLS handshake takes place to agree on algorithms, exchange cryptographic parameters and certificates, and then start the encrypted connection.

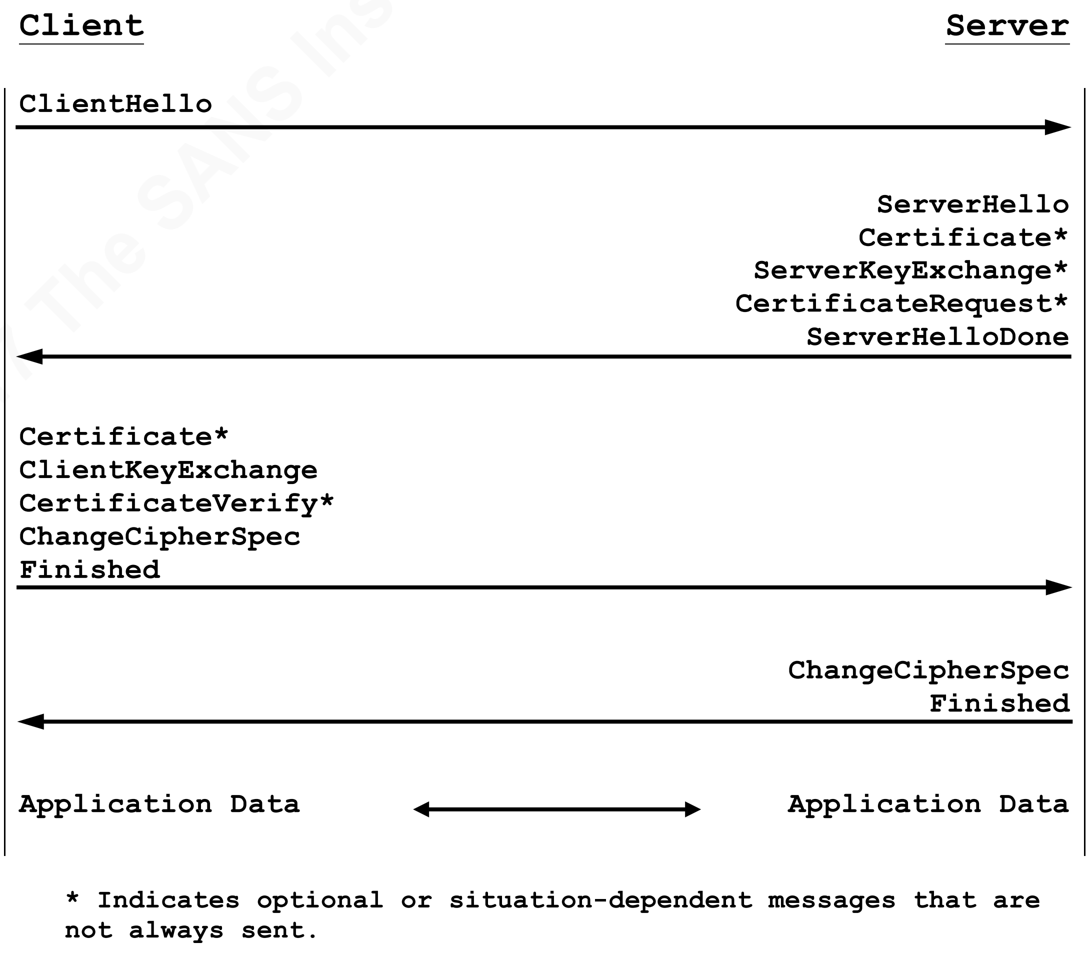


Figure 2: *TLS Full Handshake (RFC5246, 2008)*

Yousef Bakhdlaghi, bakhdlaghi@gmail.com

1.2.1. TLS Authentication and key exchange

The process of key exchange in a TLS handshake is to create a pre-master secret known to both parties (client and server) and then use it to generate the master secret. Our concern here is how this pre-master secret is shared. There are two ways to do that: RSA key exchange and Diffie-Hellman (DH) key exchange. In RSA key exchange, the pre-master secret is transmitted (encrypted) over the network. With DH, it is not transferred over the network – instead it is generated on both sides so it cannot be intercepted. However, it is possible for an analyst to decrypt TLS connections when RSA key exchange used if he has the server's private (decryption) key. But in the ephemeral form of Diffie-Hellman (DHE) key exchange, different DH keypairs will be generated for multiple handshakes, which provide the Perfect Forward Secrecy that makes the DHE highly recommended over the simple DH (RFC5246, 2008).

1.2.2. SSL/TLS Cipher Suites

A cipher suite is a combination of cryptographic algorithms that is used for key exchange, encryption, and message authentication in SSL/TLS connection as shown in *Figure 3*. Different operating systems and servers can have different cipher suites and different priority ordering (MSDN, n.d.).

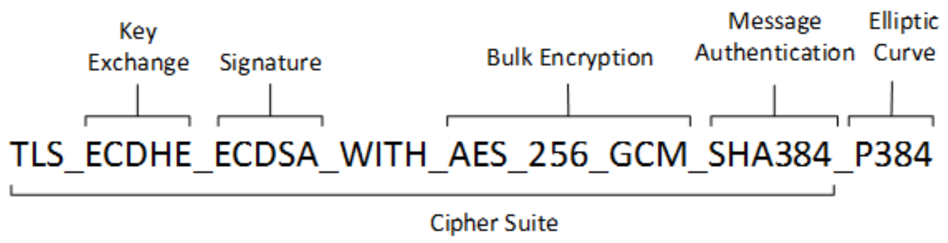


***Figure 3**: Cipher Suite (MSDN, n.d.)*

Examples of cipher suites that use RSA or DH for key exchange:

- *TLS_RSA_WITH_AES_256_CBC_SHA*
- *TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384_P384*
- *TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384_P521*

1.3. Ways to Inspect Encrypted Connections

1.3.1. Perform the inspection on the server itself

The simplest way to inspect the encrypted traffic is by employing a Host-Based IDS (HIDS) on the server itself, where the traffic belonging to that server is decrypted. An HIDS can monitor the server's activities and look for unusual behaviors, modifications to databases, system files, or any critical data. Installing the HIDS could add extra load that can negatively affect performance especially for a busy server~~.~~

1.3.2. SSL/TLS termination proxy (reverse-proxy)

A reverse-proxy is a server that acts as an intermediary between backend servers and clients. It accepts client requests and retrieves resources effectively hiding the backend servers from the clients (Villanueva, 2012). The reverse-proxy server can be configured to perform SSL/TLS encryption acting as an SSL/TLS termination proxy, which takes the load off decrypting SSL/TLS connections passing the unencrypted traffic to the associated servers. However, using an SSL/TLS termination proxy allows us to employ the IDS inside the internal network of the servers (Romero, 2016).

1.3.3. The IDS performs the decryption

In this case, the IDS is given the capability of performing the decryption process provided the private key. It could be a pre-processor or plug-in that supports decrypting and normalizing the traffic before goes to detection engine. Currently, there is no available pre-processor for Snort to perform the decryption process although it is theoretically possible to develop such pre-processor or plug-in (Snort FAQ, n.d.). However, the decryption feature is available in some propriety IDS devices like Juniper IDP (Juniper, 2013).

1.3.4. Standalone tool performs the decryption

Software or hardware that performs the SSL/TLS decryption process provided the private key then passes the decrypted traffic to the IDS. Viewssld is an example of standalone tool (free open-source) that can decrypt SSL/TLS traffic. It works by listening to an interface on a particular IP address, decrypting the encrypted traffic using the server's private key, and outputting the decrypted traffic to the listening port of the IDS.

2. Demonstration

This section demonstrates two approaches for decrypting SSL/TLS connections: termination proxy, and standalone tool to decrypt the connection.

2.1. SSL/TLS termination proxy (reverse-proxy)

Four virtual machines were used to conduct this demonstration: one Windows 7 (client), two Windows Server 2012 (SSL/TLS termination proxy server and backend server), and one Ubuntu (for sniffing and detection purposes). Also, two virtual networks were created to connect these virtual machines as shown in *Figure 4.*

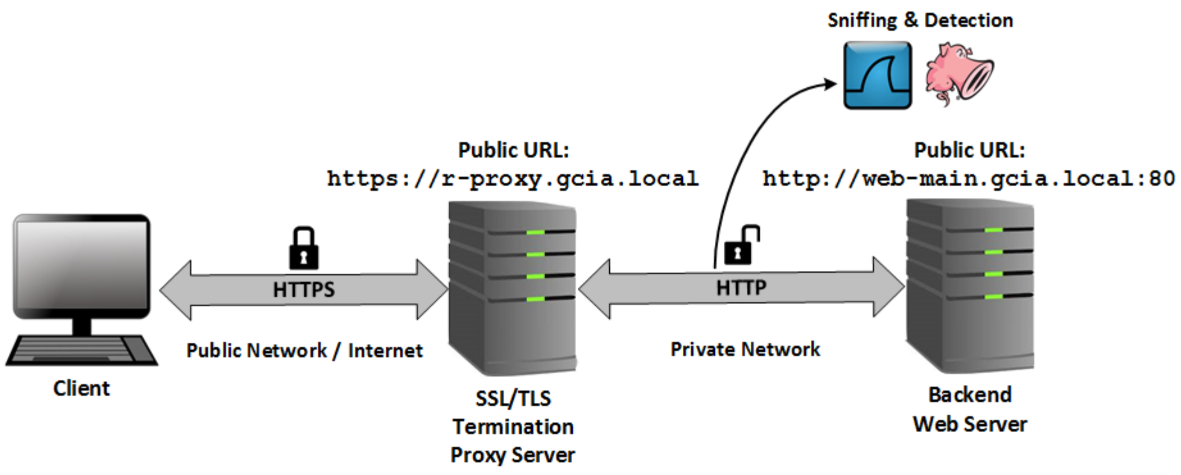

Sniffing & Detection
Public URL:
https://r-proxy.gcia.local
Public URL:
http://web-main.gcia.local:80
HTTPS
Public Network / Internet
HTTP
Private Network
Client
SSL/TLS Termination Proxy Server
Backend Web Server

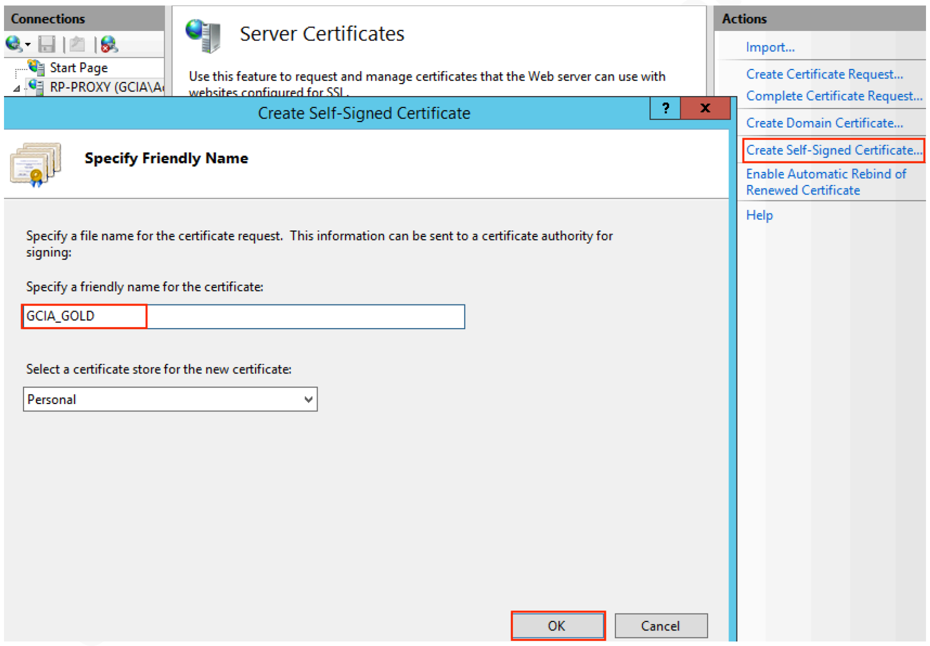
Figure 4: *Servers and virtual networks setup*

2.1.1. Server configuration

The SSL/TLS termination proxy server was configured with two interfaces: one serves clients on the public network over an SSL/TLS connection (HTTPS) and the second is connected to the backend web server over an unencrypted connection (HTTP) on the internal network. Both servers (r-proxy and web-main) were members of a domain called gcia.local. The backend server was configured as a web server running Internet Information Services (IIS) to host the site with no additional settings. In the proxy server, there are three important settings required to act as reverse-proxy: An SSL/TLS certificate (a self-signed certificate), URL Rewrite, and Application Request Routing (ARR). URL Rewrite and ARR are an extension to enable IIS to function as an SSL/TLS termination proxy (both are available through Web Platform Installer (Sfanos, 2015)).

2.1.2. Creating a self-signed RSA certificate using IIS

Creating a self-signed certificate is a built-in feature in IIS that allows issuing a self-signed certificate as shown in *Figure 5*. This feature generates an RSA certificate (2048 Bits).


Connections
Start Page
RP-PROXY (GCIA\A
Server Certificates
Use this feature to request and manage certificates that the Web server can use with websites configured for SSL.
Create Self-Signed Certificate
Specify Friendly Name
Actions
Import...
Create Certificate Request...
Complete Certificate Request...
Create Domain Certificate...
Create Self-Signed Certificate...
Enable Automatic Rebind of Renewed Certificate
Help
OK
Cancel

Specify a file name for the certificate request. This information can be sent to a certificate authority for signing:

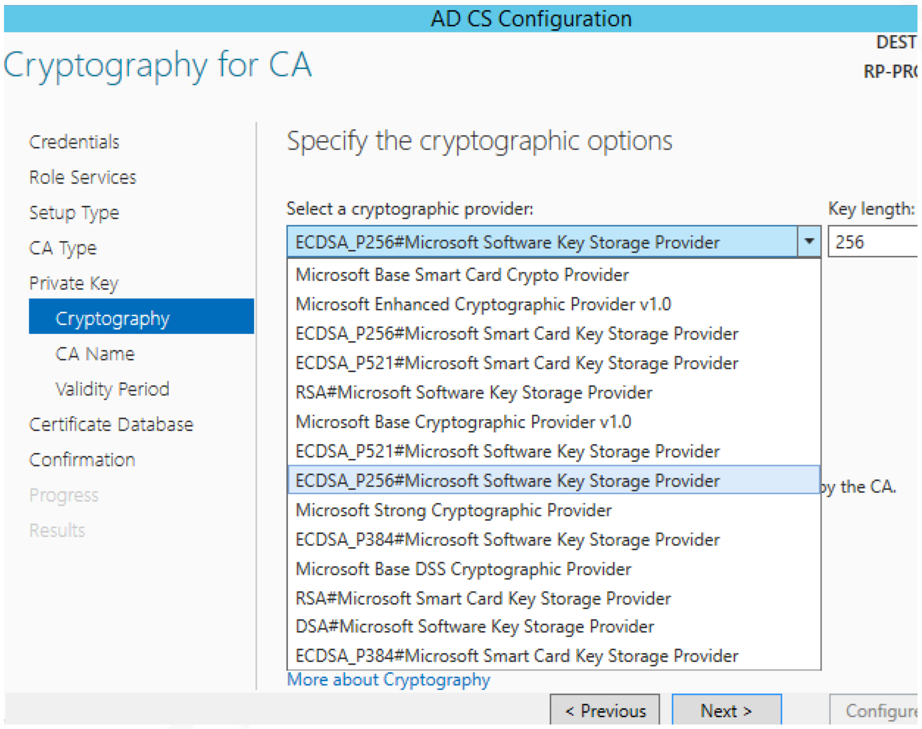
Specify a friendly name for the certificate: GCIA_GOLD

Select a certificate store for the new certificate: Personal

Figure 5: *Creating an RSA self-signed certificate*

2.1.3. Creating a self-signed ECDSA using ADCS

One way to create a self-signed ECDSA certificate to be used with the DH key exchange is through *Active Directory Certificate Services (ADCS)* role in Windows 2012 Server. After installing *ADCS*, a post-deployment configuration is required. It offers various cryptographic options to create certificates. As illustrated in *Figure 6, ECDSA_P256#Microsoft Software Key Storage Provider* was selected. At the end of this configuration, a self-signed ECDSA certificate was generated.



Specify the cryptographic options

Select a cryptographic provider: ECDSA_P256#Microsoft Software Key Storage Provider

Key length: 256

Microsoft Base Smart Card Crypto Provider
Microsoft Enhanced Cryptographic Provider v1.0
ECDSA_P256#Microsoft Smart Card Key Storage Provider
ECDSA_P521#Microsoft Smart Card Key Storage Provider
RSA#Microsoft Software Key Storage Provider
Microsoft Base Cryptographic Provider v1.0
ECDSA_P521#Microsoft Software Key Storage Provider
ECDSA_P256#Microsoft Software Key Storage Provider
Microsoft Strong Cryptographic Provider
ECDSA_P384#Microsoft Software Key Storage Provider
Microsoft Base DSS Cryptographic Provider
RSA#Microsoft Smart Card Key Storage Provider
DSA#Microsoft Software Key Storage Provider
ECDSA_P384#Microsoft Smart Card Key Storage Provider

by the CA.

More about Cryptography

Figure 6: *Creating a self-signed ECDSA certificate for DH key exchange*

2.1.4. Binding the self-signed certificate to the web site

The actions column (when the website is selected) offers a *bindings* feature that allows binding the website to a cryptographic certificate. *Figure 7* illustrates the steps to bind the website to a certificate.

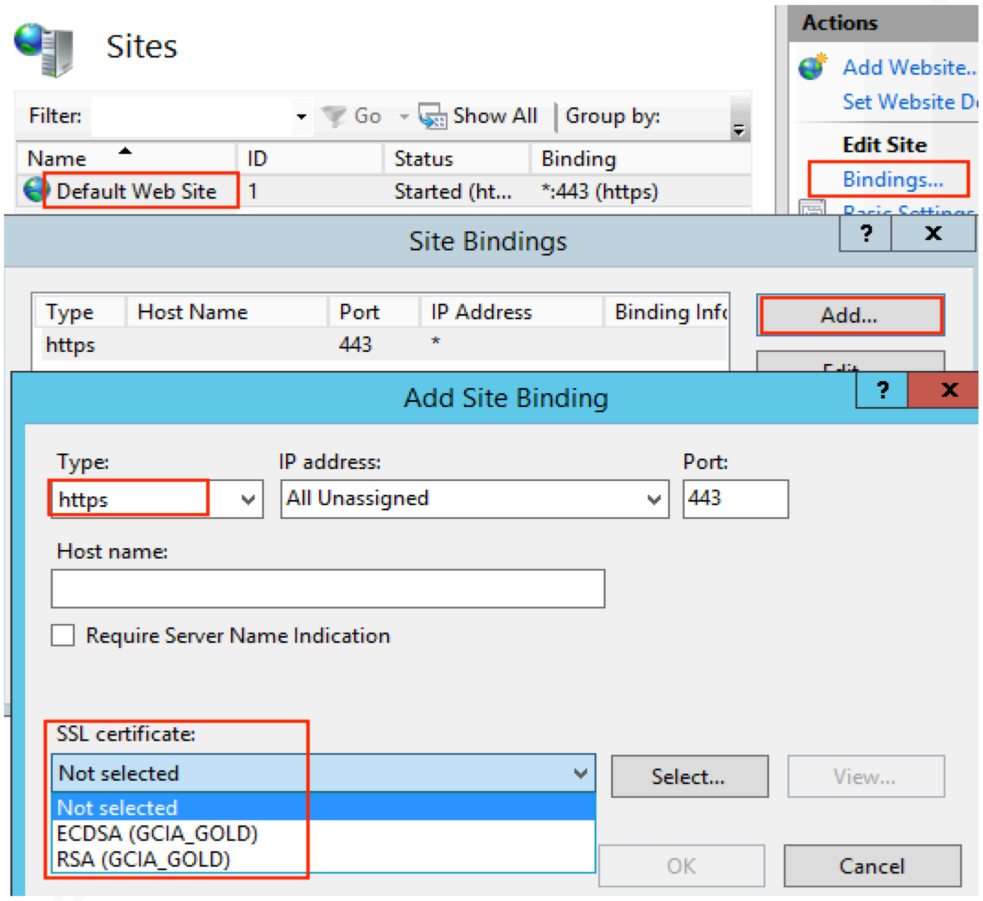
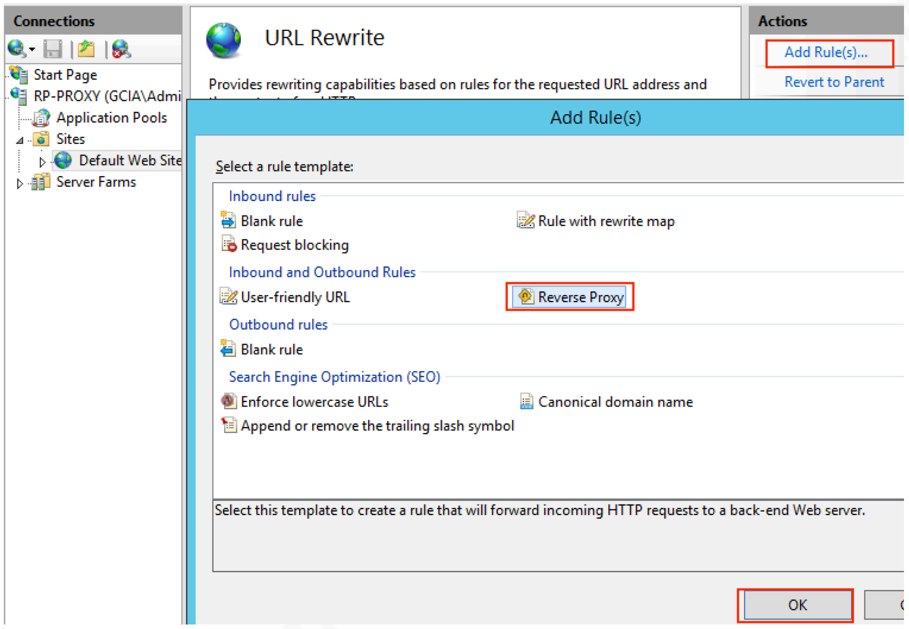


Figure 7: *Binding the cryptographic certificate to a website*

2.1.5. Configuring the SSL/TLS termination proxy server (reverse-proxy)

To add a reverse-proxy rule template, click on *Add rule(s)* from *URL Rewrite* then select *Reverse-Proxy* rule template as shown in *Figure 8*. The final step in configuring the reverse-proxy server is illustrated in *Figure 9*.

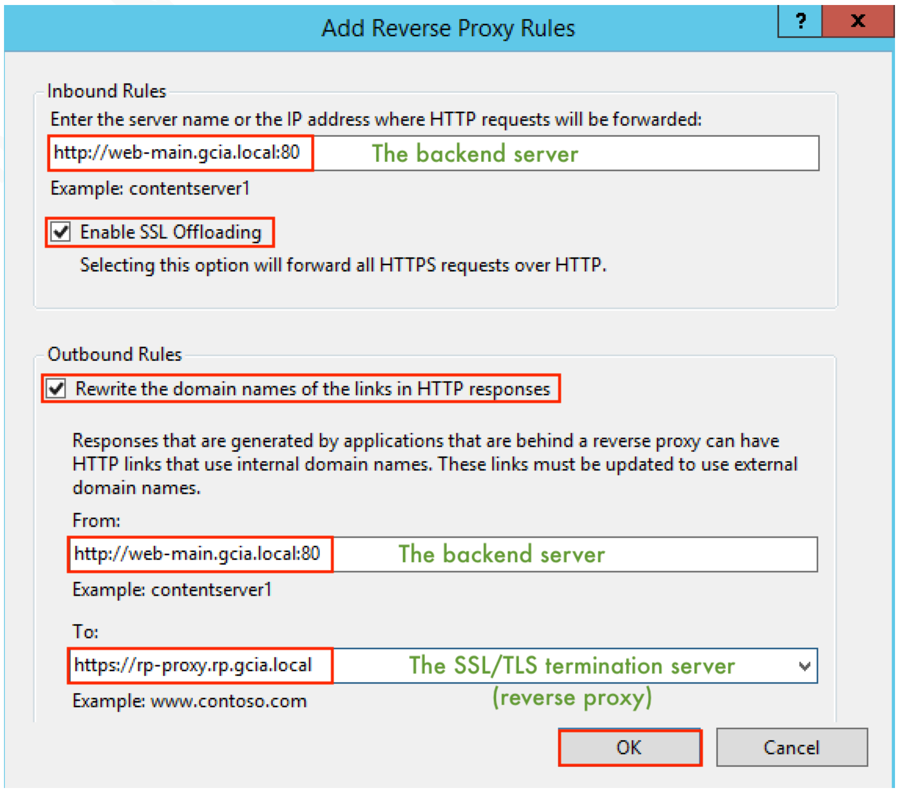


Provides rewriting capabilities based on rules for the requested URL address and

Select a rule template:

Select this template to create a rule that will forward incoming HTTP requests to a back-end Web server.

Figure 8: *Adding a reverse-proxy rule template*



Enter the server name or the IP address where HTTP requests will be forwarded:

http://web-main.gcia.local:80

Example: contentserver1

☑ Enable SSL Offloading

Selecting this option will forward all HTTPS requests over HTTP.

☑ Rewrite the domain names of the links in HTTP responses

Responses that are generated by applications that are behind a reverse proxy can have HTTP links that use internal domain names. These links must be updated to use external domain names.

From:

http://web-main.gcia.local:80

Example: contentserver1

To:

https://rp-proxy.rp.gcia.local

Example: www.contoso.com

Figure 9: *Configuring reverse-proxy rules*

Yousef Bakhdlaghi, bakhdlaghi@gmail.com

Enforcing various cipher suites

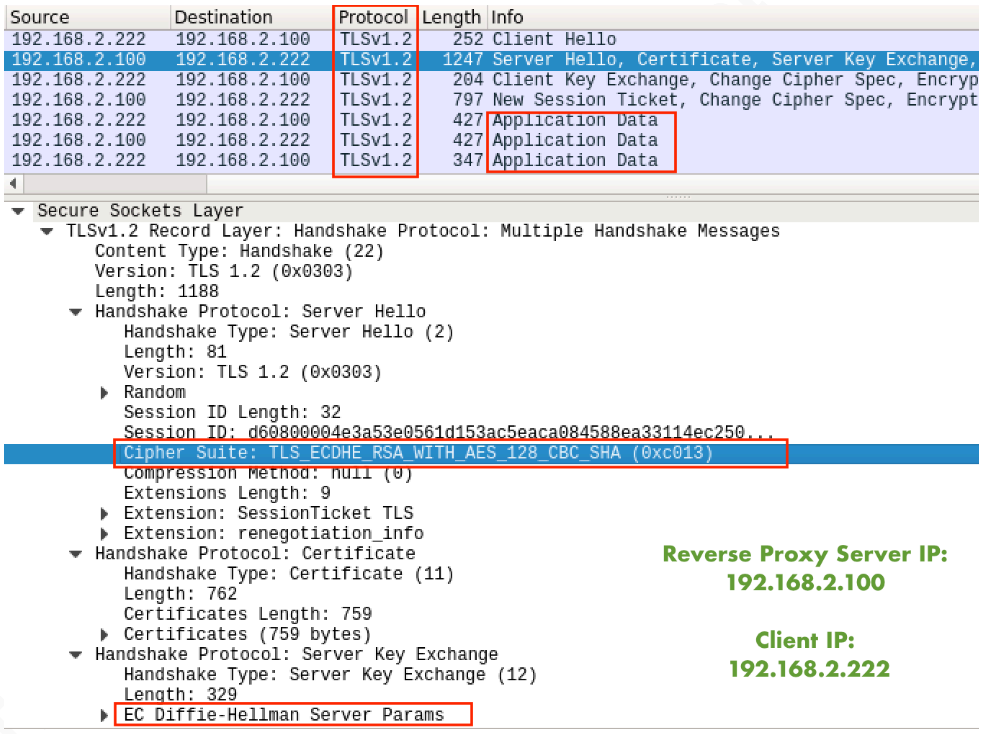
To force IIS to use specific cipher suites, the Local Group Policy Editor can be utilized (*Computer Configuration → Administrative Templates → Network → SSL Configuration Settings*). The *SSL Cipher Suite Order* needs to be enabled first, then the specific cipher suite selected. Normally, the cipher suite agreed upon by the client and server during the TLS handshake is the supported client cipher suite that ranks highest in the server's cipher suite order. For the purposes of this demonstration, the server was configured to support a single cipher suite at a time, cycling through all the entries in the client's ordered list.

Several cipher suites are tested in this demonstration; all of them are supported and decrypted by the *reverse-proxy rules* in IIS:

- *TLS_RSA_WITH_AES_128_CBC_SHA*
- *TLS_RSA_WITH_AES_256_CBC_SHA*
- *TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA*
- *TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256*
- *TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384*

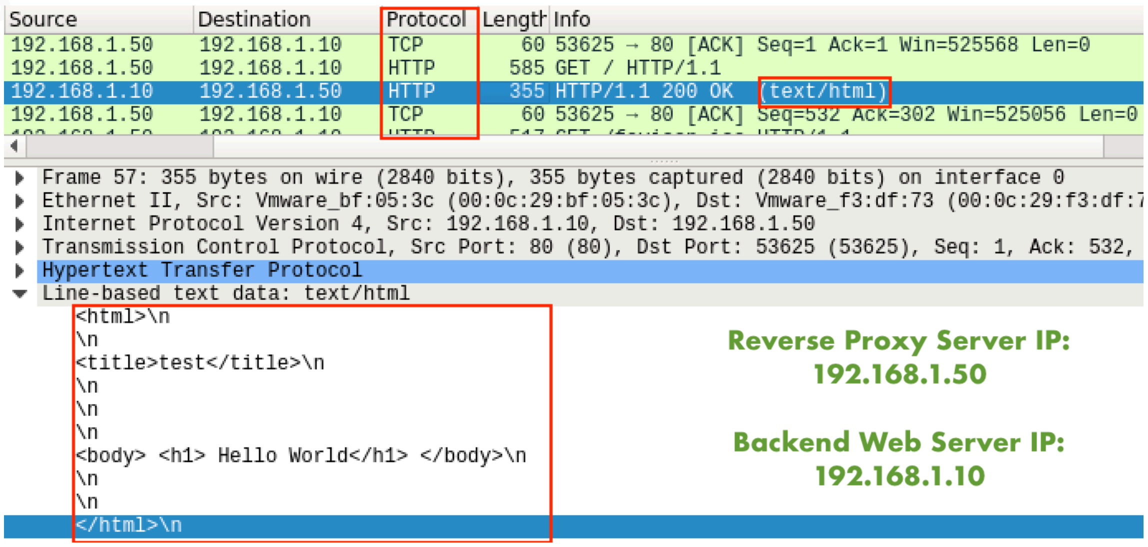
2.1.6. Sniffing the traffic on the public and the internal virtual networks

Below are screenshots captured by *Wireshark* after the connection has been made between the client and the reverse-proxy. *Figure 10* and *Figure 11* show the exchanged messages when *TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA* cipher was used.



```
Source          Destination     Protocol Length Info
192.168.2.222   192.168.2.100   TLSv1.2    252 Client Hello
192.168.2.100   192.168.2.222   TLSv1.2   1247 Server Hello, Certificate, Server Key Exchange,
192.168.2.222   192.168.2.100   TLSv1.2    204 Client Key Exchange, Change Cipher Spec, Encryp
192.168.2.100   192.168.2.222   TLSv1.2    797 New Session Ticket, Change Cipher Spec, Encrypt
192.168.2.222   192.168.2.100   TLSv1.2    427 Application Data
192.168.2.100   192.168.2.222   TLSv1.2    427 Application Data
192.168.2.222   192.168.2.100   TLSv1.2    347 Application Data

▼ Secure Sockets Layer
  ▼ TLSv1.2 Record Layer: Handshake Protocol: Multiple Handshake Messages
      Content Type: Handshake (22)
      Version: TLS 1.2 (0x0303)
      Length: 1188
    ▼ Handshake Protocol: Server Hello
        Handshake Type: Server Hello (2)
        Length: 81
        Version: TLS 1.2 (0x0303)
      ▶ Random
        Session ID Length: 32
        Session ID: d60800004e3a53e0561d153ac5eaca084588ea33114ec250...
        Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (0xc013)
        Compression Method: null (0)
        Extensions Length: 9
      ▶ Extension: SessionTicket TLS
      ▶ Extension: renegotiation_info
    ▼ Handshake Protocol: Certificate
        Handshake Type: Certificate (11)
        Length: 762
        Certificates Length: 759
      ▶ Certificates (759 bytes)
    ▼ Handshake Protocol: Server Key Exchange
        Handshake Type: Server Key Exchange (12)
        Length: 329
      ▶ EC Diffie-Hellman Server Params
```

Figure 10: *Part of a TLS handshake (TLS_ECDHE_RSA)*



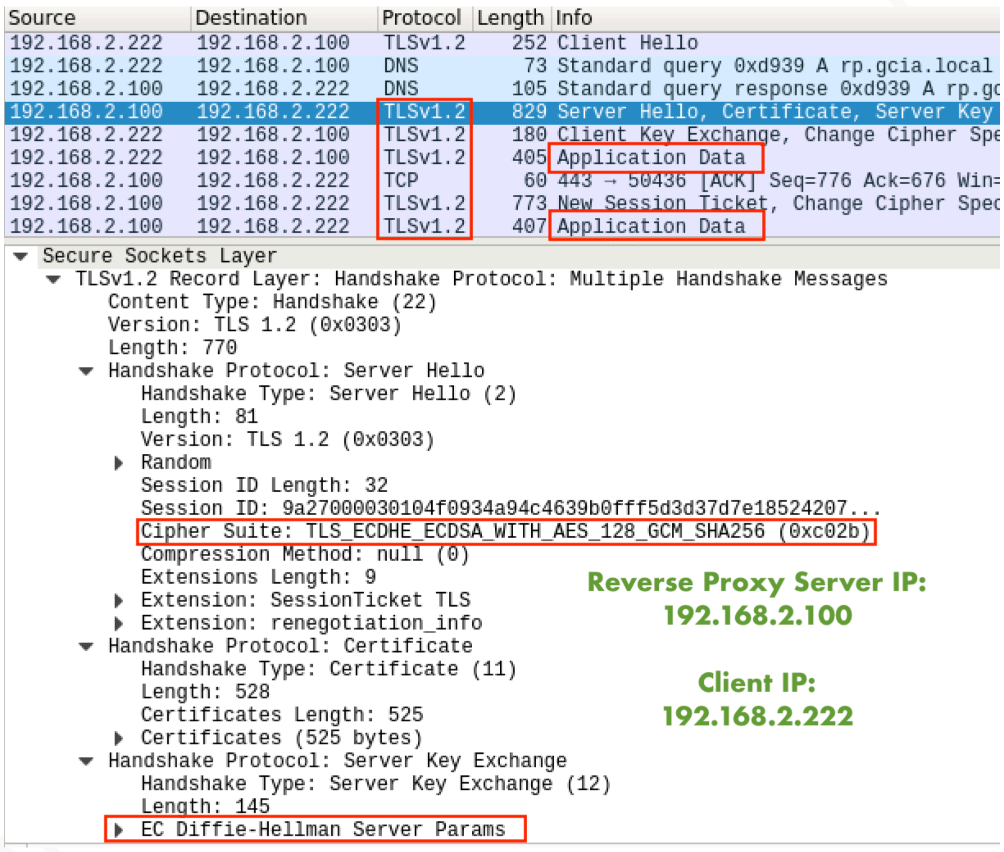
```
Source          Destination     Protocol Length Info
192.168.1.50    192.168.1.10    TCP        60 53625 → 80 [ACK] Seq=1 Ack=1 Win=525568 Len=0
192.168.1.50    192.168.1.10    HTTP      585 GET / HTTP/1.1
192.168.1.10    192.168.1.50    HTTP      355 HTTP/1.1 200 OK  (text/html)
192.168.1.50    192.168.1.10    TCP        60 53625 → 80 [ACK] Seq=532 Ack=302 Win=525056 Len=0

▶ Frame 57: 355 bytes on wire (2840 bits), 355 bytes captured (2840 bits) on interface 0
▶ Ethernet II, Src: Vmware_bf:05:3c (00:0c:29:bf:05:3c), Dst: Vmware_f3:df:73 (00:0c:29:f3:df:7
▶ Internet Protocol Version 4, Src: 192.168.1.10, Dst: 192.168.1.50
▶ Transmission Control Protocol, Src Port: 80 (80), Dst Port: 53625 (53625), Seq: 1, Ack: 532,
▶ Hypertext Transfer Protocol
▼ Line-based text data: text/html
    <html>\n
    \n
    <title>test</title>\n
    \n
    \n
    \n
    <body> <h1> Hello World</h1> </body>\n
    \n
    \n
    </html>\n
```

Reverse Proxy Server IP: 192.168.1.50
Backend Web Server IP: 192.168.1.10

Figure 11: *The HTTP response from the backend web server to the reverse-proxy (unencrypted)*

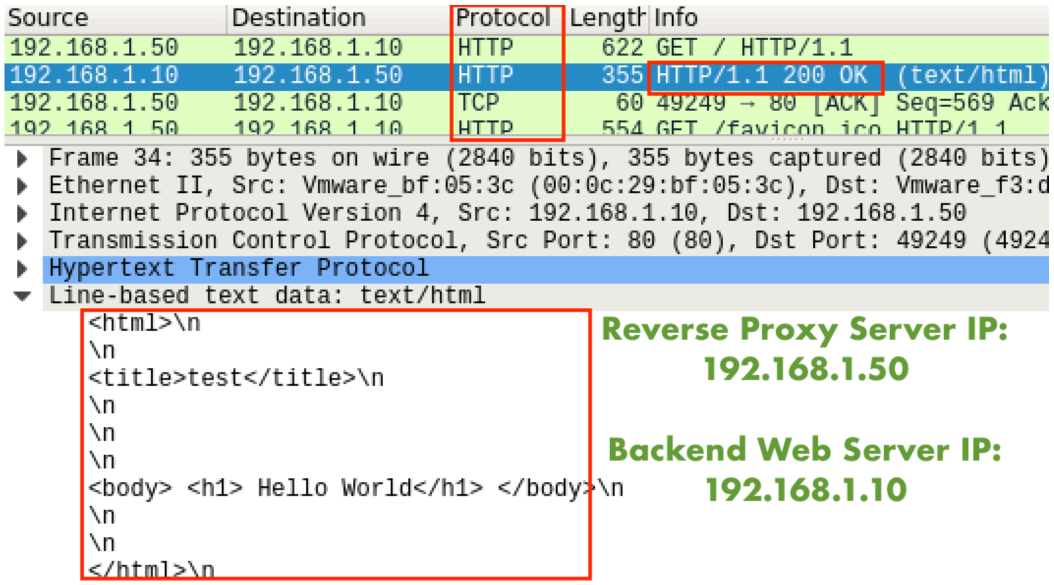
Yousef Bakhdlaghi, bakhdlaghi@gmail.com

Figure 12 and *Figure 13* show the exchanged messages when *TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256* cipher was used.



```
Source          Destination     Protocol Length Info
192.168.2.222   192.168.2.100   TLSv1.2     252 Client Hello
192.168.2.222   192.168.2.100   DNS          73 Standard query 0xd939 A rp.gcia.local
192.168.2.100   192.168.2.222   DNS         105 Standard query response 0xd939 A rp.gc
192.168.2.100   192.168.2.222   TLSv1.2     829 Server Hello, Certificate, Server Key
192.168.2.222   192.168.2.100   TLSv1.2     180 Client Key Exchange, Change Cipher Spe
192.168.2.222   192.168.2.100   TLSv1.2     405 Application Data
192.168.2.100   192.168.2.222   TCP          60 443 → 50436 [ACK] Seq=776 Ack=676 Win=
192.168.2.100   192.168.2.222   TLSv1.2     773 New Session Ticket, Change Cipher Spec
192.168.2.100   192.168.2.222   TLSv1.2     407 Application Data
▼ Secure Sockets Layer
  ▼ TLSv1.2 Record Layer: Handshake Protocol: Multiple Handshake Messages
      Content Type: Handshake (22)
      Version: TLS 1.2 (0x0303)
      Length: 770
    ▼ Handshake Protocol: Server Hello
        Handshake Type: Server Hello (2)
        Length: 81
        Version: TLS 1.2 (0x0303)
      ▶ Random
        Session ID Length: 32
        Session ID: 9a27000030104f0934a94c4639b0fff5d3d37d7e18524207...
        Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 (0xc02b)
        Compression Method: null (0)
        Extensions Length: 9
      ▶ Extension: SessionTicket TLS
      ▶ Extension: renegotiation_info
    ▼ Handshake Protocol: Certificate
        Handshake Type: Certificate (11)
        Length: 528
        Certificates Length: 525
      ▶ Certificates (525 bytes)
    ▼ Handshake Protocol: Server Key Exchange
        Handshake Type: Server Key Exchange (12)
        Length: 145
      ▶ EC Diffie-Hellman Server Params
```

Figure 12: *Part of a TLS handshake (TLS_ECDHE_ECDSA)*



```
Source          Destination     Protocol Length Info
192.168.1.50    192.168.1.10    HTTP        622 GET / HTTP/1.1
192.168.1.10    192.168.1.50    HTTP        355 HTTP/1.1 200 OK  (text/html)
192.168.1.50    192.168.1.10    TCP          60 49249 → 80 [ACK] Seq=569 Ack
192.168.1.50    192.168.1.10    HTTP        554 GET /favicon.ico HTTP/1.1
▶ Frame 34: 355 bytes on wire (2840 bits), 355 bytes captured (2840 bits)
▶ Ethernet II, Src: Vmware_bf:05:3c (00:0c:29:bf:05:3c), Dst: Vmware_f3:d
▶ Internet Protocol Version 4, Src: 192.168.1.10, Dst: 192.168.1.50
▶ Transmission Control Protocol, Src Port: 80 (80), Dst Port: 49249 (4924
▶ Hypertext Transfer Protocol
▼ Line-based text data: text/html
    <html>\n
    \n
    <title>test</title>\n
    \n
    \n
    \n
    <body> <h1> Hello World</h1> </body>\n
    \n
    \n
    </html>\n
```

Figure 13: *The HTTP response from the backend web server to the reverse-proxy (unencrypted)*

2.2. Decrypting SSL/TLS using Viewssld

As mentioned earlier, there are two ways for key exchange: RSA key exchange and DH key exchange. In RSA key exchange, a static keypair is used for the exchange and could enable a standalone tool to decrypt all the encrypted connections to that server if the private key is provided. But it is different with the ephemeral form of DH. DHE was introduced to provide Perfect Forward Secrecy. It uses different DH keypairs for multiple handshakes. Even if a DH keypair were provided, it would be possible to decrypt one connection only.

In this section, the Viewssld tool was used to decrypt an SSL/TLS connection that used RSA key exchange. Viewssld is a free open-source tool that can decrypt SSL/TLS traffic for an IDS. It works by listening to an interface on a particular IP address, decrypting the encrypted traffic using the server's private key, and outputting the decrypted traffic to the listening port of the IDS. It does not support DHE key exchange; It only supports RSA key exchange (Plashchynski, 2015).

To illustrate the process, a self-signed RSA certificate was generated using OpenSSL, then the certificate was bound to the website in Windows Internet Information Services (IIS). Two virtual machines were used to conduct this demonstration: Windows 7 (server) and Ubuntu (client). Snort and Viewssld were running on the client machine.

2.2.1. Creating a self-signed RSA certificate using OpenSSL

A self-signed RSA certificate was created using the following command in OpenSSL:

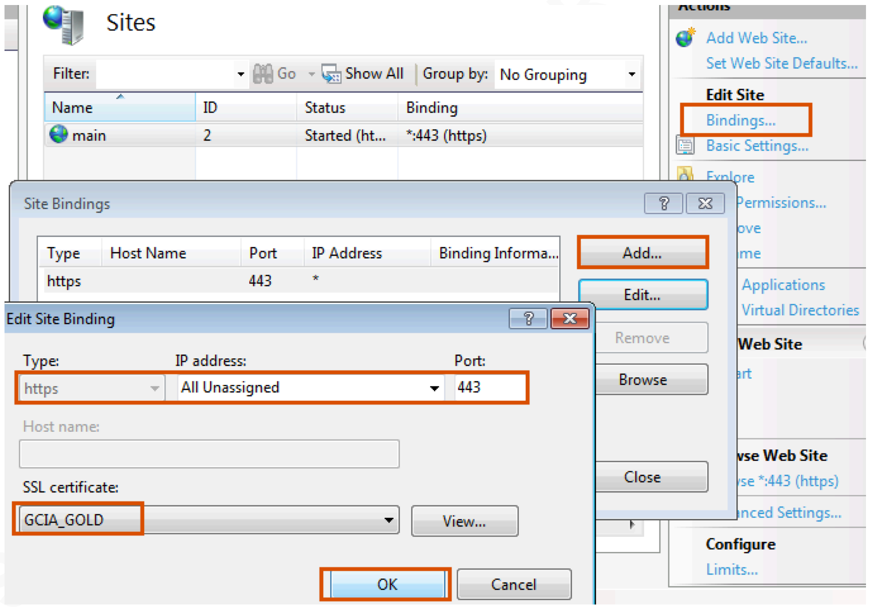
```
openssl req -x509 -nodes -newkey rsa:2048 -keyout pkey.pem -out
cert.pem
```

Then the private key and certificate were exported to pfx format to be used in Microsoft IIS:

```
openssl pkcs12 -inkey pkey.pem -in cert.pem -export -out
cert.pfx
```

2.2.2. Setting Windows IIS server

After installing IIS, a simple HTML page was created in the local site. Then, the site was bound to the self-signed RSA certificate after importing it in IIS as shown in *Figure 4*.



Name	ID	Status	Binding
main	2	Started (ht...	*:443 (https)

Type	Host Name	Port	IP Address	Binding Informa...
https		443	*	

Type: https
IP address: All Unassigned
Port: 443
Host name:
SSL certificate: GCIA_GOLD

***Figure 14:** Binding self-signed certificate to the web site*

To force Windows IIS to use specific cipher suites, the Local Group Policy Editor can be utilized (*Computer Configuration -> Administrative Templates -> Network -> SSL Configuration Settings*). The *SSL Cipher Suite Order* needs to be enabled first, then the specific cipher suite selected, namely RSA (since RSA uses the public key and private key for the key exchange). The list of the supported TLS cipher suites in Windows 7 can be obtained from (MSDN, n.d.).

2.2.3. Configuring and running Viewssld

Viewssld uses a configuration file that contains all the necessary information to run the command. Below are the settings used in the configuration file:

```
# daemonize? on/off (default: off)
daemon = off
#server1 configuration
[server1]
src = en1                #the interface where the encrypted traffic can be found
dst = en2                #destination interface for passing the decrypted traffic to
ip = 192.168.100.135     #the server's IP address to monitor
port = 443               #the port to listen on
dsslport = 80            #the destination port for the decrypted traffic
key = ~/.ssh/id_rsa/pkey.pem      #provides the private key
```

Now, we can run Viewssld using the following command:

```
sudo Viewssld --config Viewssld.conf –v
```

What follows is a sample of the output of Viewssld when it captured an SSL/TLS connection that could be decrypted:

```
=> New Session: 192.168.100.135:443<->192.168.100.133:49223
C->S: 301 bytes
S->C: 301 bytes
C->S: 302 bytes
S->C: 1382 bytes
<= Session closing: 192.168.100.135:443<->192.168.100.133:49223
pkts recv: 336 pkts drop: 0
```

2.2.4. Findings

In Windows 7 IIS, several cipher suites have been tested in this experiment, unfortunately, Viewssld was only able to decrypt one cipher suite "`TLS_RSA_WITH_RC4_128_MD5`". After checking *libdssl-master* (Viewssld dependency library), only the cipher suites below were listed in the source code and are currently supported by Viewssld.

TLS

- AES_128_CBC,SHA1
- AES_256_CBC,SHA1

SSL2

- RC4,MD5
- RC4,MD5
- RC2,MD5
- RC2,MD5
- IDEA,MD5
- DES,MD5
- SN_DES_EDE3_CBC,MD5

SSL3

- NULL,MD5
- NULL,SHA1
- RC4,MD5
- RC4,MD5
- RC4,SHA1
- RC2,MD5
- IDEA,SHA1
- DES,SHA1
- DES,SHA1
- DES3,SHA1

Unfortunately, this tool supports old cipher suites that are insecure and rarely used by servers today. However, the capability here is limited to the tool and what it supports. It is possible to enhance the open source tool's cipher suite support, but that would take development effort.

Yousef Bakhdlaghi, bakhdlaghi@gmail.com

3. Conclusion

SSL/TLS inspection is an important and desired feature for security analysts, but it has its costs. Choose the method to decrypt traffic based on the needs and the design of the network: on the server itself, an SSL/TLS termination proxy, or using a standalone tool or capability added to the IDS. If the HIDS is installed on the server itself, it could add some extra load that can negatively affect performance especially for a busy server. The standalone tool option is limited to the tool's ability and what it supports. It may be possible to enhance the tool, since its open source, but it would take an effort to develop the needed capability (cipher suite support). Also, having both encrypted and decrypted traffic could have a serious disk utilization impact, especially if a tool like Viewssld is used. Among the two options that are demonstrated here, the most feasible option is the SSL/TLS termination proxy (reverse-proxy). It can terminate the encrypted connections then pass the decrypted traffic to the associated servers over a normal HTTP connection (on the internal network). This allows the IDS to be installed and function inside the internal network.

References

Caswell, B., Beale, J., & Baker, A. (2007). *Snort IDS and IPS Toolkit (Jay Beale's Open Source Security).* SYNGRESS.

Juniper. (2010). *Inspection of SSL Traffic Overview*. Retrieved Mar 3, 2017, from Juniper Networks: https://www.juniper.net/techpubs/en_US/idp5.0/topics/concept/intrusion-detection-prevention-ssl-decryption-overview.html

Juniper. (2013). *Juniper TechLibrary: IDP SSL Overview*. Retrieved Apr 7, 2017, from Juniper Networks: https://www.juniper.net/documentation/en_US/junos12.1x44/topics/concept/idp-ssl-overview.html

Kannan, R. D. (2011). *An Experimental Study of Detecting and Correlating Different Intrusions.* SANS Reading Room.

MSDN. (n.d.). *Cipher Suites in TLS/SSL (Schannel SSP)*. Retrieved Mar 15, 2017, from Microsoft MSDN: https://msdn.microsoft.com/en-us/library/aa374757(VS.85).aspx

MSDN. (n.d.). *TLS Cipher Suites in Windows 7*. Retrieved Mar 15, 2017, from Microsoft MSDN: https://msdn.microsoft.com/en-us/library/mt767780(v=vs.85).aspx

Plashchynski, D. (2015). *viewssld README*. Retrieved 3 10, 2017, from github.com: https://github.com/plashchynski/viewssld

RFC5246. (2008). *The Transport Layer Security (TLS) Protocol Version 1.2*. Retrieved Feb 14, 2017, from The Internet Engineering Task Force (IETF): https://datatracker.ietf.org/doc/rfc5246/

RFC6176. (2011). *Prohibiting Secure Sockets Layer (SSL) Version 2.0*. Retrieved Feb 25, 2017, from Internet Engineering Task Force (IETF): https://tools.ietf.org/html/rfc6176

RFC7568. (2015). *Deprecating Secure Sockets Layer Version 3.0*. Retrieved Feb 25, 2017, from Internet Engineering Task Force (IETF): https://tools.ietf.org/html/rfc7568

Ristić, I. (2015). *BULLETPROOF SSL AND TLS: Understanding and Deploying SSL/TLS and PKI to Secure Servers and Web Applications.* Feisty Duck.

Romero, R. (2016). *Scaling Open Source Network Proxy: Leveraging Hitch For Ssl/Tls Terminations* . Retrieved Mar 14, 2017, from Varnish Software: https://info.varnish-software.com/blog/scaling-open-source-network-proxy-leveraging-hitch-ssl-tls

Sfanos, C. (2015). *Web Platform Installer Direct Downloads*. Retrieved Mar 27, 2017, from The Official Microsoft IIS Site: https://www.iis.net/learn/install/web-platform-installer/web-platform-installer-direct-downloads

Snort FAQ. (n.d.). Retrieved Nov 7, 2016, from Snort Official Website: https://www.snort.org/faq/readme-ssl

Villanueva, J. C. (2012, Aug 6). *Forward Proxy vs Reverse Proxy*. Retrieved Feb 20, 2017, from JSCAPE: http://www.jscape.com/blog/bid/87783/Forward-Proxy-vs-Reverse-Proxy