Global Information Assurance Certification Paper

Correctly Implementing Forward Secrecy

GIAC (GCIH) Gold Certification

Author: Chris Schum, cschum@centralbank.com
Advisor: Richard Carbone

Accepted: March 14, 2014

Abstract

Forward Secrecy further protects the confidentiality and integrity of the information transmitted during a TLS connection and encrypts TLS connections to a website or service.

Forward Secrecy offers substantial privacy and confidentiality benefits for encrypted channels accessing the Internet. Unfortunately, its benefits are often times not fully realized due to configuration errors, misconfiguring services that negatively affect the effectiveness of Forward Secrecy, or avoiding the use of it because of a requirement to support legacy systems that do not have the ability to utilize it. To address these issues, this paper will describe how users can implement Forward Secrecy to its full benefits.

HTTPS traffic can contain some of the most sensitive information belonging to individuals and businesses such as SSNs, account numbers, balances and user names or passwords. However, it may also contain less sensitive information, but information that one would expect to be protected from any unauthorized viewing or collection. Without correct implementation, the goal of Forward Secrecy can never be achieved and any data, whether sensitive or not, could have its privacy compromised.

1. Introduction

At the heart of Forward Secrecy is the use of the Diffie-Hellman key exchange. In addition, in order to gain the benefits of Forward Secrecy, ephemeral keys must be utilized during the exchange. OWASP describes ephemeral keys as:

> *Ephemeral keys are temporary keys used for one instance of a protocol execution and then thrown away. An ephemeral key has the benefit of providing forward secrecy, meaning a compromise of the site or service's long term (static) signing key does not facilitate decrypting past messages because the key was temporary and discarded (once the session terminated).* [1]

Therefore, to ensure that ephemeral keys are used in a Diffie-Hellman key exchange, either the Ephemeral Diffie-Hellman (DHE) or Ephemeral Elliptic Curve Diffie-Hellman (ECDHE) modes should be used. According to OpenSSL, ephemeral Diffie-Hellman keys are designed as:

> *Ephemeral Diffie-Hellman uses temporary, public keys. Each instance or run of the protocol uses a different public key. The authenticity of the server's temporary key can be verified by checking the signature on the key. Because the public keys are temporary, a compromise of the server's long term signing key does not jeopardize the privacy of past sessions. [2]*

In relation to ECDHE, Ivan Ristic's book *Bulletproof SSL and TLS* explains the ephemeral elliptic curve Diffie-Hellman key exchange in detail:

> *The ephemeral elliptic curve Diffie-Hellman (ECDHE) is conceptually similar to Diffie-Hellman, but it uses a different mathematical foundation at the core. As the name implies, ECDHE is based on elliptic curve (EC) cryptography.*
>
> *An ECDH key takes place over a specific elliptic curve, which is for the server to define. The curve takes the role of domain parameters in DH. In theory, static ECDH key exchange is supported, but in practice only the ephemeral variant (ECDHE) is used. [3]*

It should be noted that Authentication is closely tied to the key exchange process. Either RSA or ECDSA can be used for authentication however this paper will focus

specifically on RSA as ECDSA is not widely supported and requires a different SSL/TLS certificate than is likely being used by most servers.

As discussed, it is absolutely necessary that ephemeral keys are used in order for Forward Secrecy to work effectively. However, there are also several other configuration settings that must be configured for Forward Secrecy to work correctly as well. These are described in Section 2.

Thankfully, many of the configuration changes needed to enable Forward Secrecy do not incur any additional cost. Even if some of the changes do have a financial cost, it is often minimal.

2. Common Configuration Errors that Compromise Forward Secrecy

Proper configuration of Forward Secrecy will provide maximum data protection. In some instances, users may require alternative solutions depending on their unique setup.

2.1. Cipher Suite Priority

Cipher suites that support Forward Secrecy are often made available from an HTTPS server. However, if they are not prioritized correctly, Forward Secrecy will not be used.

For encrypted sessions, a browser provides a list of supported cipher suites to the web server that determines the specific suite configuration. Ideally, the server would negotiate and utilize the most secure suites. In most instances, the server selects the first cipher suite in its priority list regardless of the strength for standard browsers such as IE, Safari, Chrome, Opera, and Firefox.

The example in *Figure 1* depicts cipher suite selection by using the Qualys SSL Server Test (located at https://www.ssllabs.com/ssltest/index.html). For the unnamed domain tested, the cipher suite order shows the first one prioritized by the server is 'TLS_RSA_WITH_AES_256_CBC_SHA'.

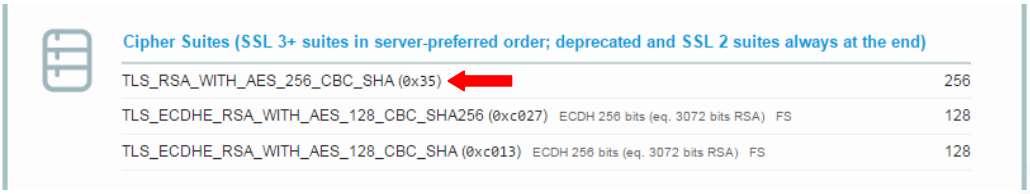

Figure 1: Prioritization Example

By using Telerik's tool 'Fiddler' (available from http://www.telerik.com/fiddler) and configuring the tool to capture SSL traffic, users can view traffic between the unnamed domain and the browser. Notice in *Figure 2* how even though the more secure ECDHE cipher suites are supported, because the AES256 suite is prioritized first, this is the cipher the will browser use.

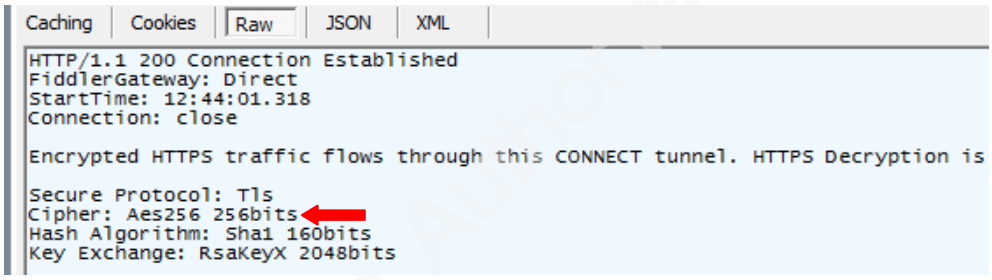

Figure 2: AES 256 Traffic Capture

By using the Qualys SSL Server Test, if the priority is re-adjusted, the negotiated cipher suites are different, as shown in *Figure 3*. With the Forward Secrecy cipher suites first, the captured traffic shows in *Figure 4* using Fiddler that traffic viewed between the unnamed domain and the browser, which demonstrates how the first Forward Secrecy cipher suite supported by the browser is in fact the first suite chosen.

Figure 3: Prioritization Change

Chris Schum, cschum@centralbank.com

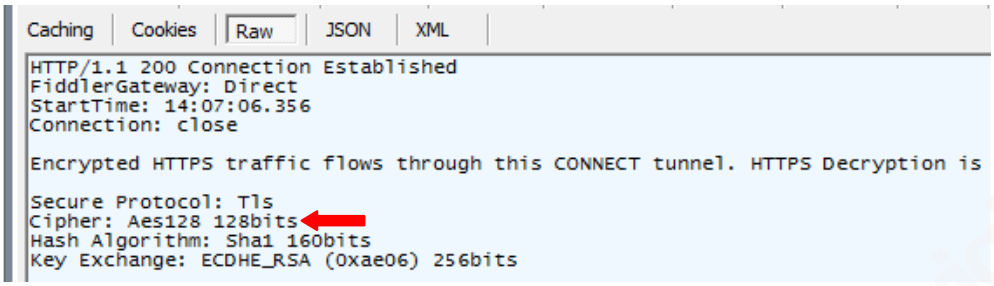

Figure 4: ECDHE Traffic Capture

Using the Qualys and Fiddler tools demonstrates that, even if a server supports cipher suites capable of Forward Secrecy, if they are not prioritized over other suites, the browser will choose whichever suite is presented first, as long as it can support said cipher suite. Forward Secrecy offers substantial security advantages to an encrypted connection and should always be prioritized first, if possible.

To ensure the highest level of session security for connecting browsers, users should prioritize cipher suites in a compatible and descending order of security. See *Figure 5* below for a visual representation of proper cipher suite negotiation between browsers and servers.

To further assist proper cipher suite prioritization and negotiation is the utilization of protocol TLSv1.2 or later. TLSv1.2 (or later) supports many secure cipher suites and enables the use of cipher suites such as Galois/Counter Mode (GCM). GCM is a block cipher, less susceptible to Cipher Block Chaining (CBC) attacks such as the Lucky13 and BEAST attacks. If TLSv1.2 or later is enabled in the browser, servers prioritize GCM capable suites. Therefore, users should enable TLSv1.2 in their browsers for proper configuration and maximum data protection.

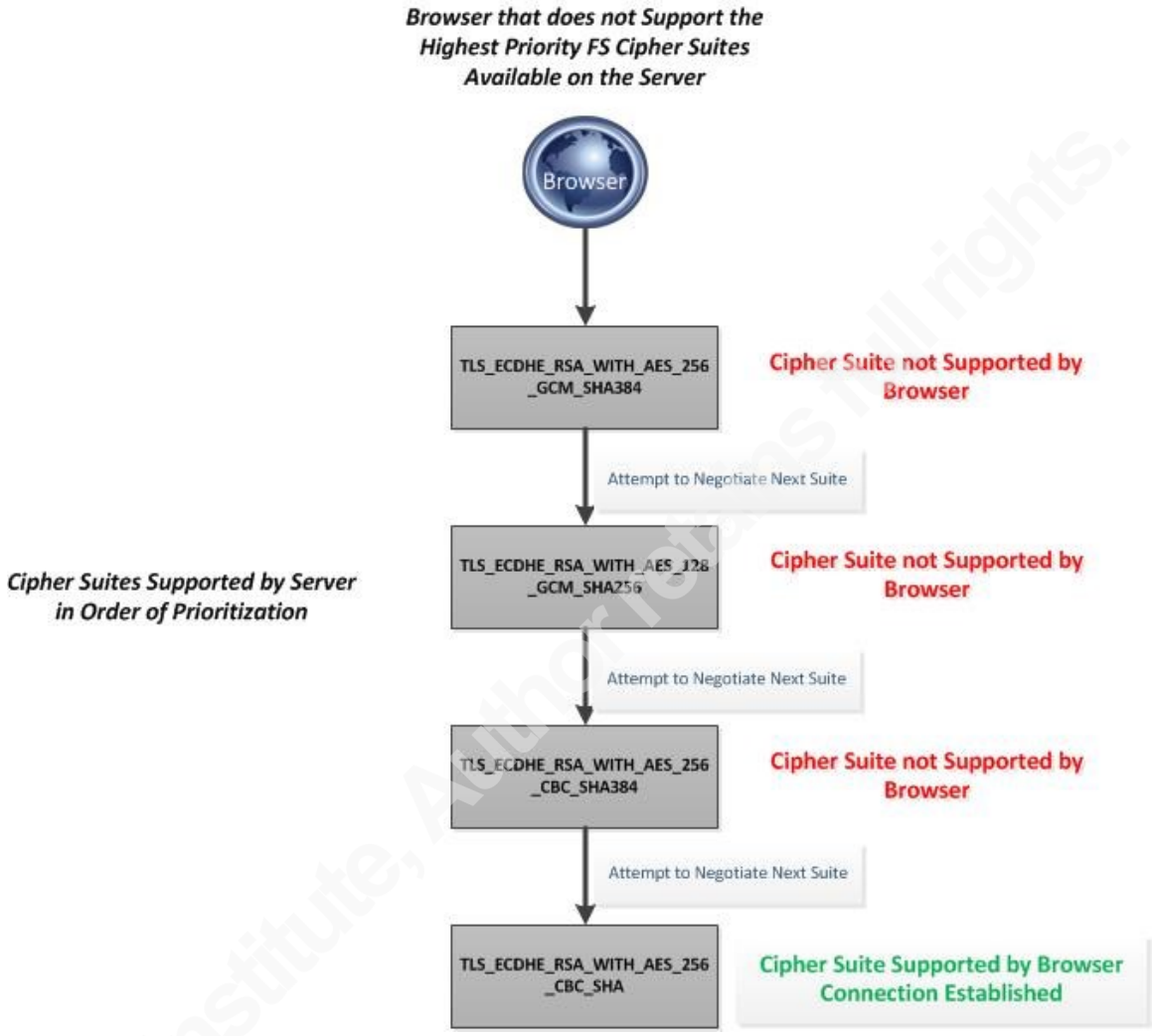

Figure 5: Browser Cipher Suite Negotiation

2.1.1 Configuring Apache for Cipher Suite Prioritization and Specific Cipher Suite Usage

Apache's cipher suite configuration directives usually reside in the 'httpd.conf' file. Depending on a system's Apache version, third party integrated Apache applications or many other scenarios, this file could be located in many different locations. Once located, the following line will need to be added to enable Cipher Suite Prioritization.

```
SSLHonorCipherOrder on
```

The aforementioned line enables the ability to prioritize the most secure cipher suites in Apache. See *Figure 5* for a visual representation of this.

Chris Schum, cschum@centralbank.com

However, the 'SSLHonorCipherOrder on' directive will only allow Apache to prioritize cipher suites. The directive for which cipher suites will be supported and in what prioritization must also be configured. There are several examples below which accomplish both these functions.

The following configurations show several options for various case scenarios. Custom configurations may be required for Apache implementations supporting legacy browsers or based on security posture.

> *Note: These configuration examples also include the removal of support for SSLv2 and SSLv3 that are deprecated and should no longer be used. They also remove support for Windows XP that is also deprecated and should not be used. If support for Windows XP is required, consider implementing SSL Offloading described in Section 3.1 of this document.*

The configuration options below are set in the Virtual Host section of the 'httpd.conf' file, beginning with '<VirtualHost *:443>' and ending with '</VirtualHost>'.

Users should first determine which cipher suites the installed version of Apache supports by using the command 'openssl ciphers –v'. After identifying the supported cipher suites, the configurations below can be adjusted to accommodate only the supported suites. If the system does not support Elliptic Curve Diffie-Hellman (ECDHE) suites, it may be beneficial to consider utilizing the SSL Offloading solution described in Section 3.1 of this document. While Ephemeral Diffie-Hellman (DHE) cipher suites support Forward Secrecy, they are much less efficient in the negotiation phase between the browser and web server and require more CPU resources.

2.1.2 Apache Forward Secrecy Cipher Suite Configuration Examples

Most Secure Long List (Only Forward Secrecy Capable Suites) – This configuration consists of a long list of cipher suites capable of Forward Secrecy but will support most browsers:

```
SSLHonorCipherOrder on
SSLCipherSuite ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-
GCM-SHA384:DHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-
```

Chris Schum, cschum@centralbank.com

```
SHA256:ECDHE-RSA-AES128-SHA:ECDHE-RSA-AES256-SHA384:ECDHE-
RSA-AES256-SHA:DHE-RSA-AES128-SHA256:DHE-RSA-AES128-
SHA:DHE-RSA-AES256-SHA256:DHE-RSA-AES256-SHA
SSLProtocol ALL -SSLv3 -SSLv2
```

Less Secure Long List (Includes non-Forward Secrecy capable Suites) – This configuration includes vast support for browsers that require some suites that do not support Forward Secrecy:

```
SSLHonorCipherOrder on
SSLCipherSuite ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-
GCM-SHA384:DHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-
SHA256:ECDHE-RSA-AES128-SHA:ECDHE-RSA-AES256-SHA384:ECDHE-
RSA-AES256-SHA:DHE-RSA-AES128-SHA256:DHE-RSA-AES128-
SHA:AES128-GCM-SHA256:AES256-GCM-SHA384:AES128-SHA:AES256-
SHA
SSLProtocol ALL -SSLv3 -SSLv2
```

Most Secure Short List (Only Forward Secrecy Capable Suites) – This configuration lists only several highly supported suites in instances where an Apache server has limited resources to assign to the cipher suite negotiation process:

```
SSLHonorCipherOrder on
SSLCipherSuite ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-
GCM-SHA384:ECDHE-RSA-AES128-SHA:ECDHE-RSA-AES256-SHA
SSLProtocol ALL -SSLv3 -SSLv2
```

Less Secure Short List (Includes non-Forward Secrecy Capable Suites) – This configuration is only necessary in limited situations where legacy browser support is needed and no other option, such as SSL Offloading is available:

```
SSLHonorCipherOrder on
SSLCipherSuite ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-
SHA:DHE-RSA-AES128-SHA:AES128-SHA
SSLProtocol ALL -SSLv3 -SSLv2
```

Optional: GSM Priority List (Includes non-Forward Secrecy Capable Suites) – This configuration could be used in the event that a vulnerability is discovered in a CBC cipher suite and/or TLSv1.0 and TLSv1.1 are somehow compromised. GCM suites could be prioritized to somewhat mitigate the vulnerability as they use TLSv1.2 and are not affected by CBC vulnerabilities. Only use this configuration in the specific scenario

above as it contains both Forward Secrecy and non-Forward Secrecy suites that could somewhat minimize the benefits:

```
SSLHonorCipherOrder on
SSLCipherSuite ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA256:AES128-GCM-SHA256:AES256-GCM-SHA384:ECDHE-RSA-AES128-SHA:ECDHE-RSA-AES256-SHA
SSLProtocol ALL -SSLv3 -SSLv2
```

2.1.3 Configuring Nginx for Cipher Suite Prioritization and Specific Cipher Suite Usage

In Nginx, the cipher suites configuration directives usually reside in the 'default.conf' file. Depending on multiple variables, that include the Nginx version, or other integrated applications, the 'default.conf' file could be located in many different locations.

The configuration options for cipher suites are typically located in the first entry in the 'default.conf' file. The file should end with a '}' as shown below:

```
server {
   listen          443;
   server_name  localhost;
}
```

The configuration options below are the same as for Apache, but have been adjusted to work with Nginx servers.

Most Secure Long List (Only Forward Secrecy Capable Suites) – This configuration consists of a long list of cipher suites capable of Forward Secrecy but will support most browsers:

```
ssl on;
ssl_certificate /etc/nginx/ssl/path/to/cert;
ssl_certificate_key /etc/nginx/ssl/path/to/key;

ssl_ciphers ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-SHA256:ECDHE-RSA-AES128-SHA:ECDHE-RSA-AES256-SHA384:ECDHE-RSA-AES256-SHA:DHE-RSA-AES128-SHA256:DHE-RSA-AES128-SHA:DHE-RSA-AES256-SHA256:DHE-RSA-AES256-SHA;

ssl_protocols TLSv1 TLSv1.1 TLSv1.2;
```

```
ssl_prefer_server_ciphers on;
```

Less Secure Long List (Includes non-Forward Secrecy capable Suites) – This configuration includes vast support for browsers that require some suites that do not support Forward Secrecy:

```
ssl on;
ssl_certificate /etc/nginx/ssl/path/to/cert;
ssl_certificate_key /etc/nginx/ssl/path/to/key;

ssl_ciphers ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-
GCM-SHA384:DHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-
SHA256:ECDHE-RSA-AES128-SHA:ECDHE-RSA-AES256-SHA384:ECDHE-
RSA-AES256-SHA:DHE-RSA-AES128-SHA256:DHE-RSA-AES128-
SHA:AES128-GCM-SHA256:AES256-GCM-SHA384:AES128-SHA:AES256-
SHA;

ssl_protocols TLSv1 TLSv1.1 TLSv1.2;

ssl_prefer_server_ciphers on;
```

Most Secure Short List (Only Forward Secrecy Capable Suites) – This configuration lists only several highly supported suites in instances where an Nginx server has limited resources to assign to the cipher suite negotiation process:

```
ssl on;
ssl_certificate /path/to/cert;
ssl_certificate_key /path/to/key;

ssl_ciphers ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-
GCM-SHA384:ECDHE-RSA-AES128-SHA:ECDHE-RSA-AES256-SHA;

ssl_protocols TLSv1 TLSv1.1 TLSv1.2;

ssl_prefer_server_ciphers on;
```

Less Secure Short List (Includes non-Forward Secrecy Capable Suites) – This configuration is only necessary in limited situations where legacy browser support is required and no other options (such as SSL Offloading) are available:

```
ssl on;
ssl_certificate /path/to/cert;
```

Chris Schum, cschum@centralbank.com

```
ssl_certificate_key /path/to/key;

ssl_ciphers ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-
SHA:DHE-RSA-AES128-SHA:AES128-SHA;

ssl_protocols TLSv1 TLSv1.1 TLSv1.2;

ssl_prefer_server_ciphers on;
```

Optional: GSM Priority List (Includes non-Forward Secrecy Capable Suites) – This configuration could be used in the event that a vulnerability is discovered in a CBC cipher suite and/or TLSv1.0 and TLSv1.1 are somehow compromised. GCM suites could be prioritized to somewhat mitigate the vulnerability as they use TLSv1.2 and are not affected by CBC vulnerabilities. Only use this configuration in the specific scenario above as it contains both Forward Secrecy and non-Forward Secrecy suites that could somewhat minimize the benefits:

```
ssl on;
ssl_certificate /path/to/cert;
ssl_certificate_key /path/to/key;

ssl_ciphers ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-
GCM-SHA384:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-
SHA256:AES128-GCM-SHA256:AES256-GCM-SHA384:ECDHE-RSA-
AES128-SHA:ECDHE-RSA-AES256-SHA;

ssl_protocols TLSv1 TLSv1.1 TLSv1.2;

ssl_prefer_server_ciphers on;
```

2.1.4 Configuring IIS for Cipher Suite Prioritization and Specific Cipher Suite Usage

When using the Windows IIS web server, the configuration directives related to cipher suites are managed by adding or removing registry entries. Tools such as 'IIS Crypto' (available at https://www.nartac.com/Products/IISCrypto/) allow users to adjust cipher suites, protocols, hashes, and key exchanges.

After users install and launch 'IIS Crypto', an interface will appear similar to *Figure 6:*

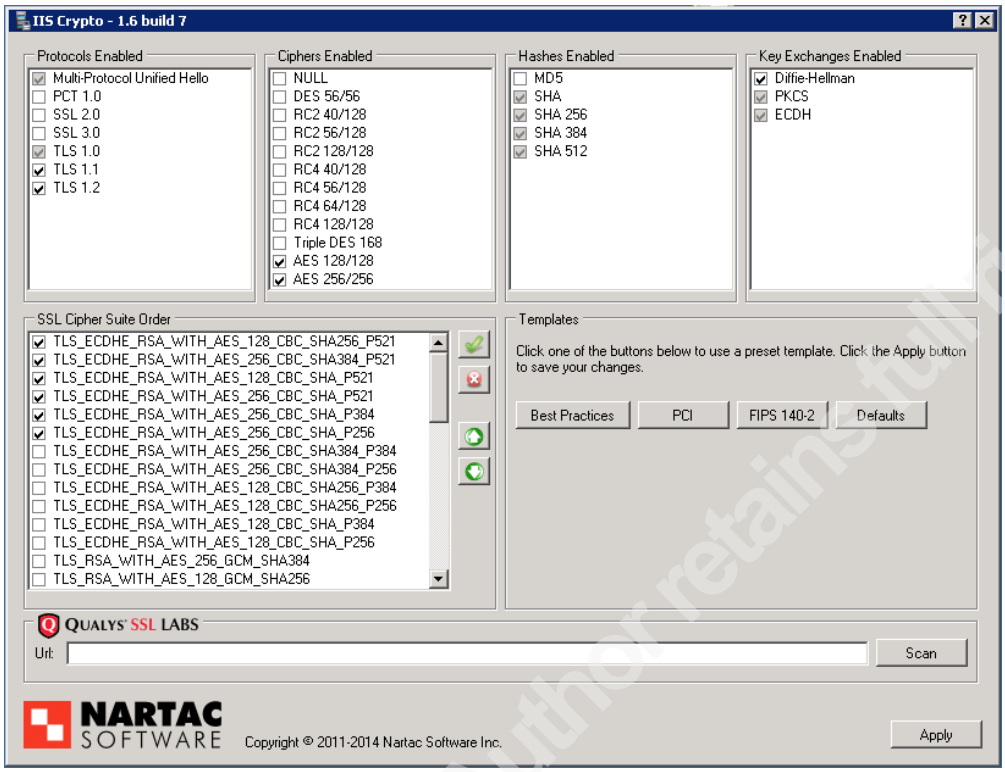

Figure 6: IIS Crypto User Interface

Users should make the following changes in order to address several security risks and enable Forward Secrecy:

Remove support for MD5 hashes by unchecking the box;

Remove support for SSL 3 and SSL 2 by unchecking the boxes;

Check only the boxes for AES 128/128 and AES 256/256 in the Ciphers Enabled section.

As it relates to ciphers, enable only those that include 'TLS_ECDHE_RSA_WITH_AES_' and prioritize them by using the green arrows to the right (see *Figure 6*).

The cipher suites utilized in *Figure 6* offer substantial security and support many browsers including all versions of Safari, Chrome, Opera, Firefox, and IE (except those used on XP).

Users on Windows Server 2003 or earlier should utilize the 'SSL Offloading' solution described in Section 3.1 because cipher suite prioritization is not available.

Chris Schum, cschum@centralbank.com

2.2. Disable SSLv3 & RC4

In order to implement Forward Secrecy, users should disable SSLv3 and RC4 cipher suites. The SSLv3 protocol and RC4 cipher suites have been insecure for some time and have cryptographic weaknesses demonstrated by the POODLE vulnerability [4].

In the event that SSLv3 or RC4 must be used for legacy systems, Section 3.1 *SSL Offloading* describes several ways that legacy systems can be supported without compromising Forward Secrecy.

2.3 TLS Session Tickets are not Refreshed Frequently

When a user initiates an HTTPS session, the browser negotiates a connection with the server determining which cipher suites, protocol, and key exchange to use. After this initial handshake, the server creates a TLS Session Ticket that is sent to the client in the event the client resumes the connection later.

TLS Session Tickets re-establish previous connections without performing another TLS handshake. This process increases performance and reduces computational overhead on the web server.

When implementing Forward Secrecy correctly, TLS Session Tickets can become troublesome because it remains in the memory of the server until the server or the HTTPS web service is restarted. Depending on how often the server or service restarts, any TLS Session Tickets may remain available in server memory. Therefore, TLS Session Tickets are a single point of compromise for any session. Section 3.2 *"Scheduled Tasks"* describes workarounds to this issue.

2.4 SSL/TLS Session IDs are not Refreshed Frequently

SSL/TLS Session IDs are similar to TLS Session Tickets and affect Forward Secrecy because Session IDs are written to a local disk on the server (depending on configuration). This could present a problem for implementing Forward Secrecy because Session IDs could be used to decrypt communications.

Chris Schum, cschum@centralbank.com

Unlike TLS Session Tickets, SSL/TLS Session IDs can be configured so that they are not written to disk, quickly refreshed in memory or both. This configuration however needs to align with business needs.

2.5 Cipher Suites Other than those Capable of Forward Secrecy are Used

Currently, there is no known attack that would cause a server supporting both Forward Secrecy cipher suites and non-Forward Secrecy cipher suites to revert or downgrade the cipher suites used. However, previous attacks such as POODLE, can downgrade cipher suites to non-Forward Secrecy suites. Limit the cipher suites supported on a web server to only those that support Forward Secrecy. Forward Secrecy supported configurations for all major browsers would then limit customer impact.

3. Alternative Solutions to Problems with Implementing Forward Secrecy

3.1 SSL Offloading

SSL Offloading manages HTTPS traffic by dedicating one server to perform all functions related to SSL/TLS encryption and SSL certificate management. In this scenario, the server(s) become the termination point for any SSL/TLS connections.

Figure 7 shows how Nginx, an open source webserver capable of SSL Offloading, displays the SSL Offloading configuration.

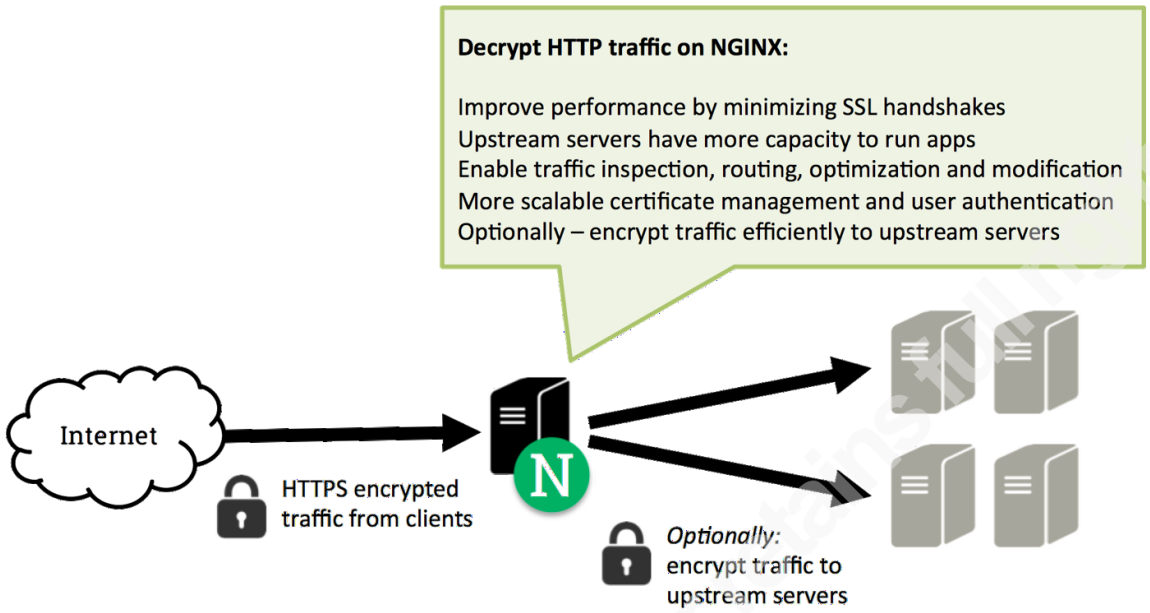

Figure 7: SSL Offloading Overview (Source: Nginx Wiki)

Nginx has a comprehensive webpage that describes in detail how to install, configure, and implement an SSL Offloading solution. The guide is available at http://wiki.nginx.org/SSL-Offloader. The Nginx server providing SSL Offloading can be configured using the 'Configuring Nginx for Cipher Suite Prioritization and Specific Cipher Suite Usage'.

This solution can address issues with legacy systems that cannot support the most secure cipher suites or cipher suite prioritization. While some configuration is required, this solution is cost effective in that Nginx is free

Another option for SSL Offloading is utilizing a Web Application Firewall, a termination point for connections to web servers that acts to offload HTTPS traffic. All offloaded traffic is scanned in real-time for web-specific attacks such as SQL Injection and Cross Site Scripting that protect servers that are protected by the Web Application Firewall. Many Web Application Firewall solutions are appliances that have the additional benefit of a user-friendly interface to manage certificates, cipher suites, hashes and key exchanges. Simultaneously, this solution protects information and is cost effective.

Note that with both the SSL Offloading solutions listed above, the issues in Section 2.3 – *'TLS Session Tickets are not Refreshed Frequently'* and Section 2.4 –

'SSL/TLS Session IDs are not Refreshed Frequently' must still be addressed as described in those sections. The SSL Offloading solutions do not address the Session Ticket and Session ID refresh issues.

3.2 Scheduled Tasks

If TLS Session Tickets or SSL/TLS Session IDs cannot be refreshed frequently, another option to mitigate the potential risk is to reboot the web server or web service on a schedule that can be established based on business needs; such as during a weekly server maintenance window. This can be done through a Scheduled Task in Windows or Cron job in Linux.

A service restart or server reboot will refresh the TLS Session Tickets or SSL/TLS Session IDs and will use those tickets and IDs for any new sessions. However, in some instances, the previous tickets or IDs may remain in server memory for an undetermined amount of time; however, this is unlikely.

4. Conclusion

Forward Secrecy offers a substantial increase in encrypted traffic protection for little or no cost. By using unique keys, information in a session can be protected even if the server is compromised in the future. Additionally, Forward Secrecy is not difficult to implement, provides substantial browser support, and requires little to no financial investment. For all these reasons, Forward Secrecy should be utilized in all possible instances to ensure the confidentially and integrity of information that passes over any network segment.

Chris Schum, cschum@centralbank.com

5. References

[1]Walton, Jeffery; Steven, John, et al. (August 14, 2014). Certificate and Public Key Pinning. Web article. Retrieved January 15, 2015. https://www.owasp.org/index.php/Certificate_and_Public_Key_Pinning#Ephemeral_Keys

[2] OpenSSL Wiki. (2015). OpenSSL: Diffie Hellmann. Retrieved January 21, 2015. http://wiki.openssl.org/index.php/Diffie_Hellman

[3] Ristic, Ivan. (2014). *Bulletproof SSL and TLS* London. Book. ISBN: 978-1907117. UK Feisty Duck.

[4] Google. (2014, October 14). Online Security Blog. Retrieved January 2015. http://googleonlinesecurity.blogspot.com/2014/10/this-poodle-bites-exploiting-ssl-30.html

Chris Schum, cschum@centralbank.com

6. Bibliography

Bernat, V. (2012, January 1). SSL/TLS & Perfect Forward Secrecy. Web article Retrieved March 5, 2015, from http://vincent.bernat.im/en/blog/2011-ssl-perfect-forward-secrecy.html

Elliptic Curve Cryptography. (2013, March 12). OpenSSL: Elliptic Curve Cryptography. Wiki. Retrieved March 4, 2015, from http://wiki.openssl.org/index.php/Elliptic_Curve_Cryptography

Google. (2014, October 14). Online Security Blog. Retrieved January 2015. http://googleonlinesecurity.blogspot.com/2014/10/this-poodle-bites-exploiting-ssl-30.html

Graham-Cumming, J. (2013, July 12). Staying on top of TLS attacks. Web article. Retrieved March 5, 2015, from https://blog.cloudflare.com/staying-on-top-of-tls-attacks/

Hoffman-Andrews, J. (2013, November 22). Forward Secrecy at Twitter. Web article. Retrieved March 2, 2015, from https://blog.twitter.com/2013/forward-secrecy-at-twitter

Langley, A. (2013, June 27). How to Botch TLS Forward Secrecy. Blog. Retrieved March 2, 2015, from https://www.imperialviolet.org/2013/06/27/botchingpfs.html

OpenSSL Wiki. (2015). OpenSSL: Diffie Hellmann. Retrieved January 21, 2015. http://wiki.openssl.org/index.php/Diffie_Hellman

Ristic, I. (2013, August 5). Configuring Apache, Nginx, and OpenSSL for Forward Secrecy. Blog. Retrieved December 8, 2014, from http://blog.ivanristic.com/2013/08/configuring-apache-nginx-and-openssl-for-forward-secrecy.html

Ristic, I. (2013, June 25). SSL Labs: Deploying Forward Secrecy. Blog. Retrieved December 8, 2014, from

https://community.qualys.com/blogs/securitylabs/2013/06/25/ssl-labs-deploying-forward-secrecy

Ristic, Ivan. (2014). *Bulletproof SSL and TLS* London. Book. ISBN: 978-1907117. UK Feisty Duck.

SSL-Offloader. (2012, April 21). Web article. Retrieved March 2, 2015, from http://wiki.nginx.org/SSL-Offloader

Wikipedia. (2014, April 9). Forward Secrecy. Online encyclopedic entry. Retrieved March 2, 2015, from http://en.wikipedia.org/wiki/Forward_secrecy

Zhu, Y. (2014, April 8). Why the Web Needs Perfect Forward Secrecy More Than Ever. Web article. Retrieved February 2, 2015, from https://www.eff.org/deeplinks/2014/04/why-web-needs-perfect-forward-secrecy

Chris Schum, cschum@centralbank.com