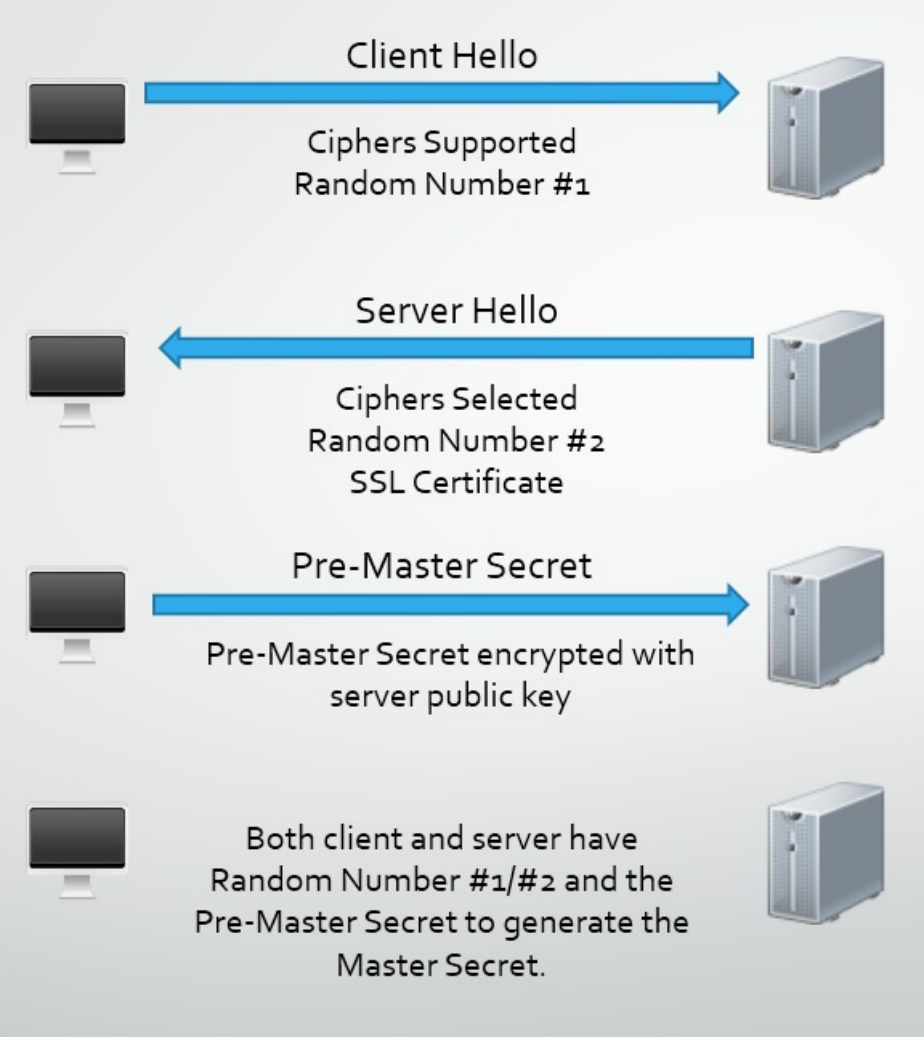
Perfect Forward Secrecy is a feature of specific key agreement protocols that gives assurances your session keys will not be compromised even if the private key of the server is compromised. By generating a unique session key for every session a user initiates, even the compromise of a single session key will not affect any data other than that exchanged in the specific session protected by that particular key. Perfect Forward Secrecy represents a huge step forwards in protecting data on the transport layer and following on from Heartbleed, everyone using SSL/TLS should be looking to implement it.

What happened before Perfect Forward Secrecy?

Prior to the implementation of PFS, all data transmitted between a server and a client could be compromised if the server's private key was ever disclosed. In particular, an attacker could record encrypted traffic for any amount of time, and store it until such a time that they had access to the private key. Once they have access to the private key, they can decrypt all historic data. This is possible because of the way that key material is exchanged between the client and the server. During the initial handshake, the client creates something called a Pre-Master Secret. The Pre-Master Secret is encrypted with the server's public key and sent to the server to protect it from being exposed whilst in transit. Once the server receives the PMS it can decrypt it with its own private key and then both client and server have their own copy. From this, both client and server generate the symmetric sessions keys that will be used to exchange further data, the Master Secret. Because the Pre-Master Secret is encrypted with the server's public key, exposure of the private key would allow an attacker to decrypt the data at any point in the future. This ability to decrypt historic data at any point represents quite a serious potential problem.



If all the traffic has been recorded by an attacker, they have Random Number #1 and #2 as they are sent in plain text, along with the Pre-Master Secret encrypted with the server public key. Once the attacker has the server private key, they can decrypt the Pre-Master Secret and generate the Master Secret to decrypt the session data.

How does Perfect Forward Secrecy help?

To enable PFS, the client and the server have to be capable of using a cipher suite that utilises the Diffie-Hellman key exchange. Importantly, the key exchange has to be

ephemeral

. This means that the client and the server will generate a new set of Diffie-Hellman parameters for each session. These parameters can never be re-used and should never be stored, the ephemeral part, and that's what offers the protection going forwards. Because of the magic behind

Diffie-Hellman

, the exchange of key material can take place in clear text without compromising the generation of a shared secret. The math is a little too heavy to detail in this blog post but there is a great example

here

on Wikipedia. Because the shared secret, the session key, is derived from complex mathematical operations carried out on the numbers exchanged between the client and the server, that are too difficult for an attacker to brute force, the attacker can at no point record data that is used to derive the session key. Even if the private key of the server is compromised, it doesn't aid the attacker in decrypting any data because the associated public key was never used to protect anything. What's even better is that with Perfect Forward Secrecy, the server generates a new set of Diffie-Hellman parameters for each session and both parties create a new shared secret that is unique and unknown to an attacker. Even if the attacker managed to compromise this shared secret somehow, it would only compromise that particular session. No previous or future sessions would be compromised.

1. Alice and Bob agree to use a prime number $p = 23$ and base $g = 5$.
2. Alice chooses a secret integer $a = 6$, then sends Bob $A = g^a \bmod p$
   - $A = 5^6 \bmod 23$
   - $A = 15{,}625 \bmod 23$
   - $A = 8$
3. Bob chooses a secret integer $b = 15$, then sends Alice $B = g^b \bmod p$
   - $B = 5^{15} \bmod 23$
   - $B = 30{,}517{,}578{,}125 \bmod 23$
   - $B = 19$
4. Alice computes $s = B^a \bmod p$
   - $s = 19^6 \bmod 23$
   - $s = 47{,}045{,}881 \bmod 23$
   - $s = 2$
5. Bob computes $s = A^b \bmod p$
   - $s = 8^{15} \bmod 23$
   - $s = 35{,}184{,}372{,}088{,}832 \bmod 23$
   - $s = 2$
6. Alice and Bob now share a secret (the number **2**).

How do I get Perfect Forward Secrecy?

Enabling support for Perfect Forward Secrecy on your server is actually fairly straight forward. Whilst the appropriate ciphers have been available since SSLv3, it's best to ensure you have support for the latest TLSv1.2 protocol. After that, it's just a case of picking the correct cipher suites and ensuring that they are ordered correctly. Any Diffie-Hellman key exchange will provide you with Forward Secrecy, but you should only select Ephemeral key exchange to obtain Perfect Forward Secrecy (a brand new session key for every session). This is usually displayed in the cipher suite in the form of DHE or EDH. You should also include Elliptic Curve DHE suites as they are faster than their DHE counterparts and should be prioritised above them where possible. You can opt to exclude DHE suites and just stick with ECDHE suites. See my

Qualys SSL Test

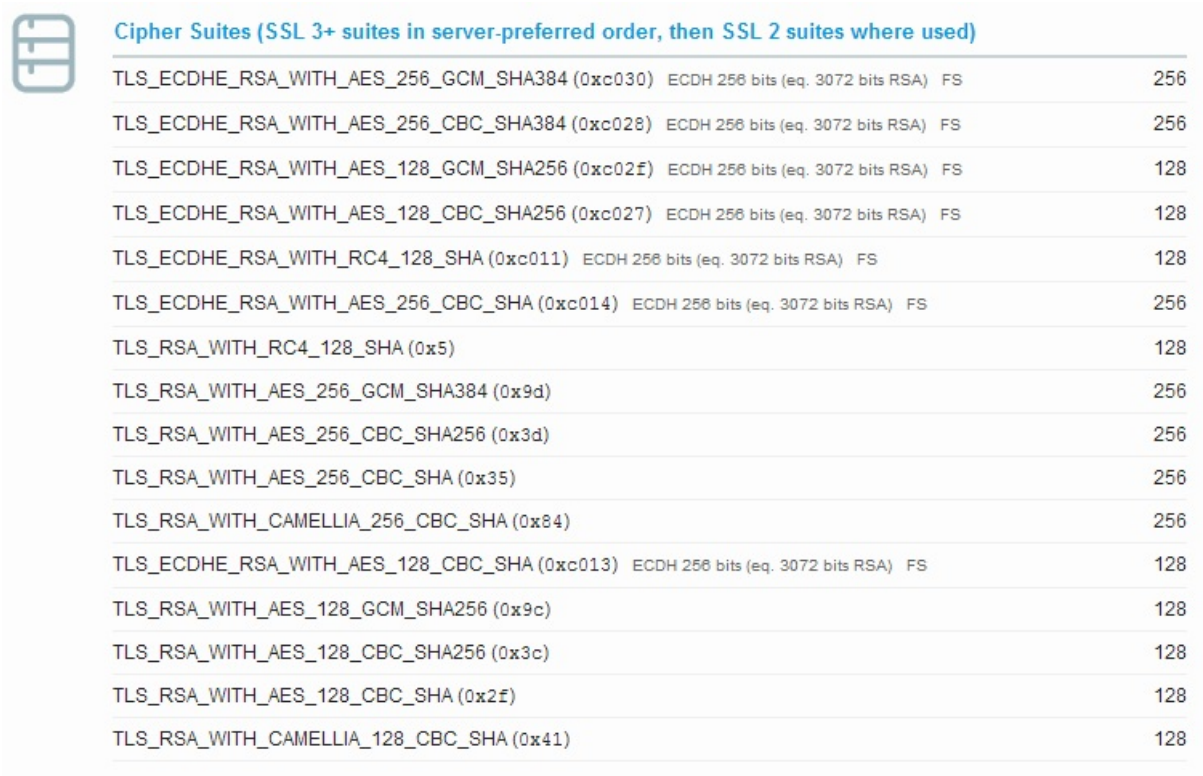
for details on which suites I'm running on my blog. Also, ensure you enforce the ordering of your ciphers by using '

ssl_prefer_server_ciphers on;'

in nginx and '

SSLHonorCipherOrder on'

in Apache.



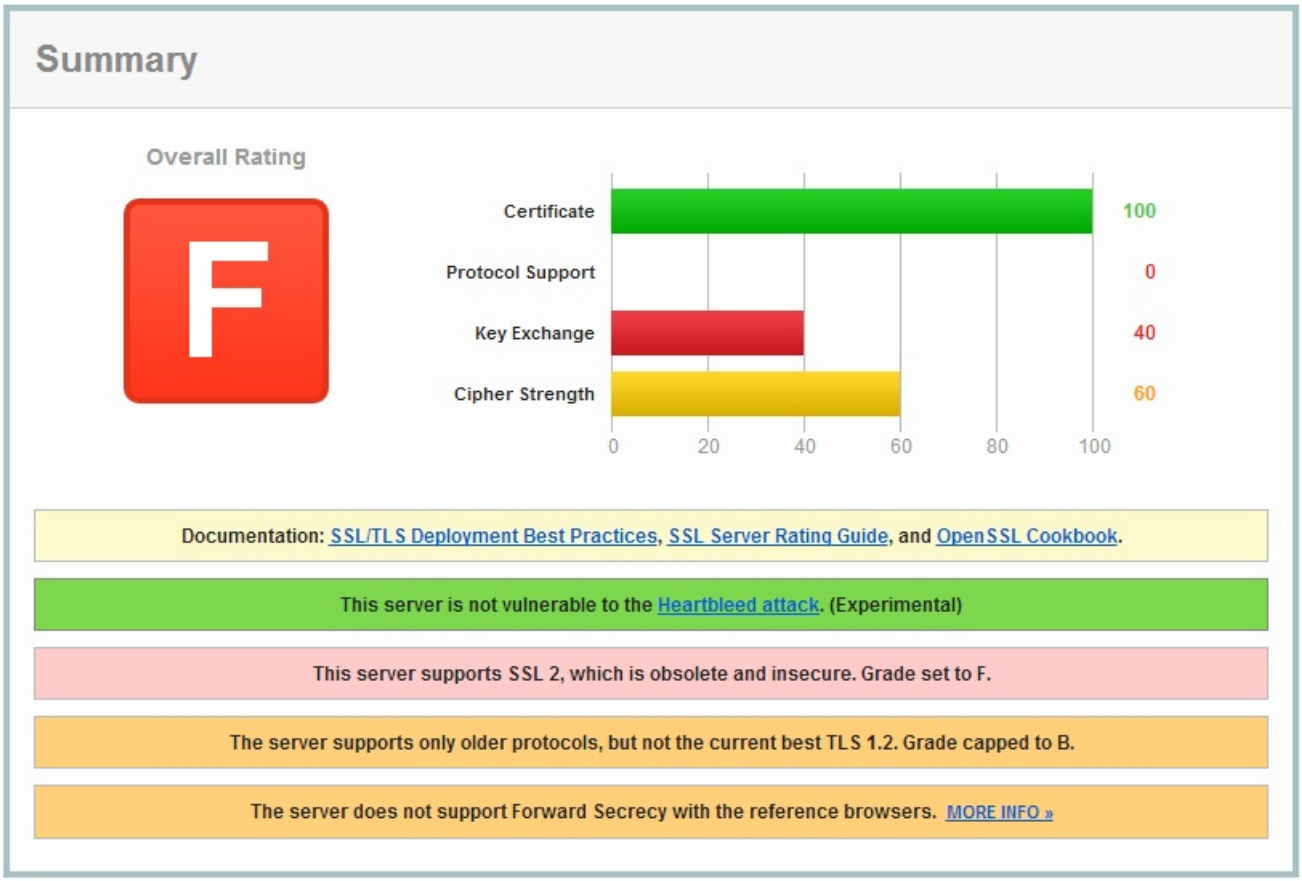
Cipher Suites (SSL 3+ suites in server-preferred order, then SSL 2 suites where used)	
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xc030) ECDH 256 bits (eq. 3072 bits RSA) FS	256
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 (0xc028) ECDH 256 bits (eq. 3072 bits RSA) FS	256
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f) ECDH 256 bits (eq. 3072 bits RSA) FS	128
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 (0xc027) ECDH 256 bits (eq. 3072 bits RSA) FS	128
TLS_ECDHE_RSA_WITH_RC4_128_SHA (0xc011) ECDH 256 bits (eq. 3072 bits RSA) FS	128
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014) ECDH 256 bits (eq. 3072 bits RSA) FS	256
TLS_RSA_WITH_RC4_128_SHA (0x5)	128
TLS_RSA_WITH_AES_256_GCM_SHA384 (0x9d)	256
TLS_RSA_WITH_AES_256_CBC_SHA256 (0x3d)	256
TLS_RSA_WITH_AES_256_CBC_SHA (0x35)	256
TLS_RSA_WITH_CAMELLIA_256_CBC_SHA (0x84)	256
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (0xc013) ECDH 256 bits (eq. 3072 bits RSA) FS	128
TLS_RSA_WITH_AES_128_GCM_SHA256 (0x9c)	128
TLS_RSA_WITH_AES_128_CBC_SHA256 (0x3c)	128
TLS_RSA_WITH_AES_128_CBC_SHA (0x2f)	128
TLS_RSA_WITH_CAMELLIA_128_CBC_SHA (0x41)	128

Why don't more sites use Perfect Forward Secrecy?

Whilst it's true that a vast majority of sites still don't implement Perfect Forward Secrecy, it seems that this is simply down to a lack of someone implementing it. Yes, Perfect Forward Secrecy does have computational overheads in generating truly ephemeral session keys, but not to the extent that hosts shouldn't be able to handle it. Much like the situation that I covered in

my blog on HSTS

, which only requires a simple, almost insignificant, HTTP response header, we haven't seen anything close to widespread adoption. If server admins aren't going to enable HSTS to enforce the use of SSL/TLS on their secure servers, we're even less likely to see them going out of their way even more to enable Perfect Forward Secrecy. It seems that as long as the user gets a padlock icon in their address bar, most hosts are happy to continue on as they are even if they are using SSLv2, 40 bit ciphers and have no preference on cipher suite ordering. The following Qualys SSL Test result may look appalling, but the user will still get a padlock in their address bar indicating that everything is just fine.



What does this have to do with Heartbleed?

The

Heartbleed

bug allows an attacker to extract data from the memory of a vulnerable server. This includes things like usernames, passwords, email addresses, session data and worst of all, the server's private key. Once the private key has been compromised, there are 2 big problems. First up, an attacker can impersonate the server by presenting the certificate that it now has the private key for. This problem can be dealt with by revoking the certificate in question so that browsers know not to trust it. Secondly, they can intercept and decrypt traffic. If an attacker has been recording encrypted traffic and storing it, once they gain access to the private key, they can go back and decrypt all of the data that they have intercepted in the past. So, let's say someone like the NSA or GCHQ has been recording the traffic to and from my blog for the last year, Heartbleed comes along and gives them access to my private key, it's game over. All that historic data is now vulnerable. This is what Perfect Forward Secrecy prevents. Because the data used to generate each session key is never actually sent over the transport layer, even with access to the private key, there isn't actually any useful information for an attacker to decrypt. They will still have to break each individual session key for each individual session, which, as it stands right now, is an insurmountable task.

Conclusion

Whilst there is a minor overhead introduced with the addition of Perfect Forward Secrecy, it's well worth taking for all the benefits it brings. You can see my blog on how to achieve an

A+ rating on the Qualys SSL Test

for more information on cipher suites and how to set it all up in nginx on Ubuntu. If you have a real requirement to run SSL on your website, then Perfect Forward Secrecy should be a no brainer.