SSL DOES NOT MEAN SOL
What if you don’t have the server keys?

J. Scott Haugdahl
Architect, Blue Cross Blue Shield MN

Robert Bullen
Systems Engineer, Blue Cross Blue Shield MN
Setting Expectations

- This session is not about..
  - An introduction to SSL encryption
  - How to set up SSL decryption in Wireshark
  - A detailed walk through of the SSL handshake and all the variants

- This session is about…
  - What you can do when you do not have access to server keys
    - Calculating server command response time from SSL, even in the cloud
    - Using encrypted data to your advantage
    - Identifying application layer behavior based on SSL patterns
  - Walking through real world examples using Wireshark
  - Focus will be on helping you to analyze application performance more so than security breaches, suspicious activity, etc.
A (very) Brief History of Secure Sockets Layer (SSL)

- Used to encrypt + protect integrity of network data
  - SSL 2.0 was first “public release” in 1995
  - SSL 3.0 released in 1996 forming the foundation for Transport Layer Security (TLS) 1.0 (RFC 2246, 1999)
  - TLS 1.0 is not backward compatible with SSL 3.0!
  - Upgraded to TLS 1.1 (RFC 4346, 2006) and TLS 1.2 (RFC 5346, 2008)

- Supports a wide variety of encryption algorithms
  - RSA and DSA are asymmetric (public key encrypts; private key decrypts) – used to exchange and generate key information during the SSL handshake
  - AES and 3DES are symmetric algorithms (one key encrypts and decrypts) – used to transfer data (much faster to compute) after the SSL handshake

- TLS 1.0 or higher is recommended practice
  - Many clients & systems now support TLS 1.2 which addresses some vulnerabilities
What’s so Special About the Client Key Exchange?

Both Client and Server generate the master secret from the pre-master to generate the session key.

Therefore, Wireshark needs the server’s private key to decrypt the client pre-master secret to order to generate the master secret to generate the session key to decrypt the SSL packet data!
# A Tale of Two Connections

Good, we will get the client key exchange!

Rats, the client is reusing a previous session ID and the server accepts.
What if we don’t have the client key exchange*?

- If your SSL session reused the Session ID…
  - Try to find a trace containing the original handshake containing the key exchange and pre-pend it

- Use Fiddler or similar
  - As a proxy that runs on the client
  - As a proxy on another workstation & point the remote client to it

- Use client pre-master secret logged by Chrome or Firefox + Wireshark
  - This is cool ‘cuz we don’t need the server key to decrypt it

- When all else fails…
  - Use knowledge of TCP & SSL segmentation to watch for inefficiencies
    - SSL payload size (small is probably ok for SSH but not FTP)!
  - Identify unlike flows across firewalls using encrypted data pattern matching
  - Look for other factors that throttle throughput in other sessions

*Or the client key exchange uses Diffie-Hellman in which we are toast even if we possess the server key.
Diffie-Hellman

- Described in a 1976 White Paper by Whitfield Diffie and Martin Hellman
- Protects against long-term key compromise (i.e. server keys!)
- Is not SSL specific, can be used for any secret information exchange
- Client generates a random number, as does server
  - Thus forms a way for the client to encrypt the pre-master (already encrypted with the server’s public key) back to the server
Diffie-Hellman
Use Case: Firewall Pattern Matching

- Perimeter firewalls NAT from private to public IP
  - Terminates TCP but maintain SSL session data
  - Unfortunately, we cannot say the same for proxy servers, load balancers, or anything else that terminates SSL connections
- Simply grab some binary data (i.e. encrypted) from SSL on one side of the firewall and filter on it to find the other side
- Once you have a match, you can then filter on the TCP streams and determine the firewall delay and other characteristics
  - Do not use SPANs nor multiple sniffers due to delays and timestamp synchronization
  - Best practice is to use taps above and below the firewall that feed a common sniffer or are combined via a visibility fabric (Apcon, Big Switch, Gigamon, Ixia, VSS, etc.)
- Also works great for following encrypted VMWare VDI streams (filter on UDP payload) across multiple tiers
Using Wireshark to Find NATed SSL Flows

1. Start with a pool of packets captures inside and outside of the firewall...

2. Filter on some SSL data from the flow of interest into the firewall...
   Which picks up the matching flow on the other side of the firewall.

3. We now have our two flows either side of the firewall for focused analysis.
Use Case: Slow eMail Migration

- Migrating user’s mailboxes from internal Lotus Notes servers to Microsoft Office 365 in the Cloud
  - Typical mailbox size was 50 GB
  - Throughput varied from 200-500 kbps over a 1 gig Internet pipe
  - 4k users @ 1 hour per user = 166 days!
- Subsequent web proxy bypass did not help nor did moving to DMZ
- Graphing the I/O revealed a potential problem area
Use Case: Slow eMail Migration

- A pattern emerged when walking through the SSL & checking neighboring flows
  - A second flow (in red below) running was clearly controlling the throughput
  - The throttling was set to approximate three bursts or blocks of data per second
  - Properties could not be changed, i.e. they are controlled by the (MS) cloud server
Use Case: Slow eMail Migration

- Each data stream was equated to one piece of mail
  - Due to control channel, conversion rate was approximately three emails per second (!)
  - Another potential optimization was to increase the application layer block size to greater than 12k (which we derived from the SSL segment size of 4112 bytes x 3 per turn)

- Solution was to run multiple servers simultaneously with multiple mailbox migrations per server to the cloud, which is per MS recommendation
  - We were running up to 40 migrations in parallel at the peak
  - All mailboxes were migrated in under 30 days
Wrapping it Up

- First gain a solid understanding of the general application layer command-response characteristics in the unencrypted world (HTTP, SQL, mail, etc.)
  - Pretend that the SSL layer *is* the application layer and apply those characteristics
- Figure out who is the client and who provides the data
  - Usually the client opens the connection, but not always!
- Breakdown the TCP segmentation and the SSL segmentation
  - Ensure that the SSL segment size makes sense for the application (SSH vs. HTTPS for instance)
- Identifying network from back-end response time is easier but must use patterns and neighboring flows for more complex cases
Thank You!

Contact us!

scott.haugdahl@bluecrossmn.com

robert.bullen@bluecrossmn.com