

SHARKFEST 2015

WIRESHARK DEVELOPER AND USER CONFERENCE



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

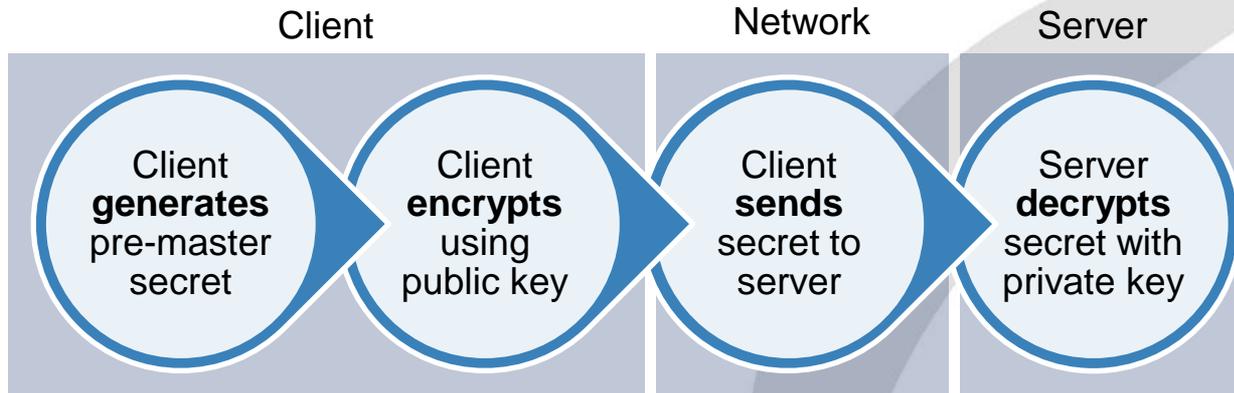
- This session is not about..
 - An introduction to SSL encryption
 - How to set up SSL decryption in Wireshark
 - A detailed walk through of the SSL handshake and all the variants
- This session is about...
 - What you can do when you *do not have access to server keys*
 - Calculating server command 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!

No.	Length	Source	Destination	Protocol	Stream index	Sess ID Len	Info
155	62	BCBSMN911	proxy-mb.	TCP	9		54785[E]9119 [SYN, Seq=312491139 Win=8192 Len=0 MSS=1460 SACK_PERM=1
156	62	proxy-mb.	BCBSMN911	TCP	9		9119[E]54785 [SYN, ACK] Seq=523685150 Ack=312491140 Win=14600 Len=0 MS
157	54	BCBSMN911	proxy-mb.	TCP	9		54785[E]9119 [ACK] Seq=312491140 Ack=523685151 Win=64240 Len=0
159	62	BCBSMN911	proxy-mb.	TCP	10		54789[E]9119 [SYN, Seq=2035020690 Win=8192 Len=0 MSS=1460 SACK_PERM=1
161	62	proxy-mb.	BCBSMN911	TCP	10		9119[E]54789 [SYN, ACK] Seq=3851424993 Ack=2035020691 Win=14600 Len=0
162	54	BCBSMN911	proxy-mb.	TCP	10		54789[E]9119 [ACK] Seq=2035020691 Ack=3851424994 Win=64240 Len=0
170	292	BCBSMN911	proxy-mb.	HTTP	9		CONNECT encrypted-tbn3.gstatic.com:443 HTTP/1.1
171	270	BCBSMN911	proxy-mb.	HTTP	10		CONNECT ssl.gstatic.com:443 HTTP/1.1
172	60	proxy-mb.	BCBSMN911	TCP	9		9119[E]54785 [ACK] Seq=523685151 Ack=312491378 Win=15544 Len=0
174	60	proxy-mb.	BCBSMN911	TCP	10		9119[E]54789 [ACK] Seq=3851424994 Ack=2035020907 Win=15544 Len=0
194	192	proxy-mb.	BCBSMN911	HTTP	10		HTTP/1.1 200 Connection established
195	268	BCBSMN911	proxy-mb.	TLSv1.2	10	0	Client Hello
196	192	proxy-mb.	BCBSMN911	HTTP	9		HTTP/1.1 200 Connection established
197	571	BCBSMN911	proxy-mb.	TLSv1.2	9	32	Server Hello
210	1514	proxy-mb.	BCBSMN911	TLSv1.2	10		Server Hello
211	504	proxy-mb.	BCBSMN911	TCP	10		[TCP segment of a reassembled PDU]
212	54	BCBSMN911	proxy-mb.	TCP	10		54789[E]9119 [ACK] Seq=2035021121 Ack=3851427042 Win=64240 Len=0
215	1514	proxy-mb.	BCBSMN911	TCP	10		[TCP segment of a reassembled PDU]
216	598	proxy-mb.	BCBSMN911	TLSv1.2	10		Certificate
217	54	BCBSMN911	proxy-mb.	TCP	10		54789[E]9119 [ACK] Seq=2035021121 Ack=3851429046 Win=64240 Len=0
219	60	proxy-mb.	BCBSMN911	TCP	9		9119[E]54785 [ACK] Seq=523685289 Ack=312491895 Win=16616 Len=0
221	207	proxy-mb.	BCBSMN911	TLSv1.2	9	32	Server Hello, Change Cipher Spec, Hello Request, Hello Request
224	316	BCBSMN911	proxy-mb.	TLSv1.2	10		Client Key Exchange, Change Cipher Spec, Hello Request, Hello Request
226	241	BCBSMN911	proxy-mb.	TLSv1.2	9		Change Cipher Spec, Hello Request, Hello Request
227	60	proxy-mb.	BCBSMN911	TCP	9		9119[E]54785 [ACK] Seq=523685442 Ack=312492082 Win=17688 Len=0
238	60	proxy-mb.	BCBSMN911	TCP	10		9119[E]54789 [ACK] Seq=3851429046 Ack=2035021383 Win=17688 Len=0
241	98	proxy-mb.	BCBSMN911	TCP	10		[TCP segment of a reassembled PDU]
242	152	proxy-mb.	BCBSMN911	TLSv1.2	9		Application Data, Application Data
243	386	proxy-mb.	BCBSMN911	TLSv1.2	10		New Session Ticket
244	54	BCBSMN911	proxy-mb.	TCP	10		54789[E]9119 [ACK] Seq=2035021383 Ack=3851429422 Win=63864 Len=0
252	54	BCBSMN911	proxy-mb.	TCP	9		54785[E]9119 [ACK] Seq=312492082 Ack=523685540 Win=63851 Len=0

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  even if we possess the server key.

Search This!

SANS Institute
InfoSec Reading Room

This paper is from the SANS Institute Reading Room site. Reposting is not permitted without express written permission.

SSL/TLS: What's Under the Hood

Encrypted data, by definition, is obscured data. Most web application authentication happens over HTTPS, which uses SSL/TLS for encryption. Did you ever wonder what that authentication exchange looks like in plaintext? What if you are troubleshooting your HTTPS enabled web application and need to dig deeper down in the OS model than Fiddler or other web developer tools will allow? This paper demonstrates how to easily decrypt and dissect a captured web session without either a proxy middleware or possession of the server...



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...

No.	Length	Delta	Time	Source	Destination	Protocol	Flow	Info
110236	70	0.000006	0.789264	edge-t1ver	172.26.35.	TCP	1067 80	[128283 [ACK] Seq=622377982 Ack=451736317 Win=28160 Len=0
110237	324	0.000003	0.789267	132.245.20		TLSv1	748	Application Data, Application Data
110238	70	0.000001	0.789268	edge-t1ver	172.26.35.	TCP	1067 80	[128283 [ACK] Seq=622377982 Ack=451737349 Win=30976 Len=0
110259	83	0.000015	0.789283	pvlaedv06.	pados02.bc	TCP	16	59012[14330 [PSH, ACK] Seq=476172060 Ack=3654822663 Win=501
110260	308	0.000022	0.789305	outlook-ma	VDITSA2003	TLSv1	1991	Application Data, Application Data
110261	64	0.000021	0.789316	VDITSA2003		TLSv1	2034	[TCP segment of a reassembled PDU] Continuation
110262	88	0.000015	0.789351	VDIDEV1067		TPKT	2024	Continuation
110263	1514	0.000001	0.789354	pod51035.o	mbintsr1-e	TLSv1	669	Application Data
110264	1514	0.000012	0.789366	pod51035.o	mbintsr1-e	TCP	669	[TCP segment of a reassembled PDU]
110265	210	0.000009	0.789375		68.178.80.	ESP		ESP (SPI=0x45887635)

2 Filter on some SSL data from the flow of interest into the firewall...

Filter: ssl.app_data == 0d674388003e968ddb47c9e27df613209ea89637e8269d5f...

No.	Length	Delta	Time	Source	Destination	Protocol	Flow	Info
110263	1514	0.000000	0.789354	pod51035.outlook.com	mbintsr1-e0-	TLSv1	669	Application Data
110269	1518	0.000066	0.789420	pod51035.outlook.com	BCBSMN89651.	TLSv1	671	Application Data

Frame 110263: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits) on interface...
Ethernet II, Src: ..., Dst: Cisco_43:0f:00 (78:da:6e:43:0f:00)
Internet Protocol Version 4, Src: pod51035.outlook.com (157.56.238.114), Dst: mbintsr1-e0-...
Transmission Control Protocol, Src Port: 443 (443), Dst Port: 49386 (49386), Seq: 546113597, Ack: 995087239, Len: 1460
Secure Sockets Layer
TLSv1 Record Layer: Application Data Protocol: http
Content Type: Application Data (23)
Version: TLS 1.0 (0x0301)
Length: 32
Encrypted Application Data: 0d674388003e968ddb47c9e27df613209ea89637e8269d5f...

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

Filter: tcp.stream == 669 or tcp.stream == 671

No.	Length	Delta	Time	Source	Destination	Protocol	Flow	Info
65689	1518	0.000013	0.473692	pod51035.out	BCBSMN89651.	TCP	671	[TCP segment of a reassembled PDU]
65949	1514	0.000767	0.474459	pod51035.out	mbintsr1-e0-	TCP	669	[TCP segment of a reassembled PDU]
65959	1518	0.000068	0.474527	pod51035.out	BCBSMN89651.	TCP	671	[TCP segment of a reassembled PDU]
65960	1514	0.000005	0.474532	pod51035.out	mbintsr1-e0-	TCP	669	[TCP segment of a reassembled PDU]
65973	1518	0.000065	0.474597	pod51035.out	BCBSMN89651.	TCP	671	[TCP segment of a reassembled PDU]
66104	1514	0.000627	0.475224	pod51035.out	mbintsr1-e0-	TCP	669	[TCP segment of a reassembled PDU]
66105	1514	0.000012	0.475236	pod51035.out	mbintsr1-e0-	TCP	669	[TCP segment of a reassembled PDU]
66113	1518	0.000053	0.475289	pod51035.out	BCBSMN89651.	TCP	671	[TCP segment of a reassembled PDU]
66115	1518	0.000012	0.475301	pod51035.out	BCBSMN89651.	TCP	671	[TCP segment of a reassembled PDU]
66251	64	0.000818	0.476119	BCBSMN89651.	pod51035.out	TCP	671	49386[1443 [ACK] Seq=995086195 Ack=545992591 Win=1790 Len=0
66252	60	0.000003	0.476122	mbintsr1-e0-	pod51035.out	TCP	669	49386[1443 [ACK] Seq=995086195 Ack=545992591 Win=1790 Len=0
66260	1514	0.000056	0.476178	pod51035.out	mbintsr1-e0-	TCP	669	[TCP segment of a reassembled PDU]
66275	1518	0.000065	0.476243	pod51035.out	BCBSMN89651.	TCP	671	[TCP segment of a reassembled PDU]

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

No.	Length	Delta	Time	Source	Destination	Protocol	Info
27	64	0.000033000	0.054483000	Proxy	MigrationServer	TCP	9119[62830]62830 [ACK] Seq=1843722766 Ack=525495198 Win=3006 Len=0
28	1518	0.000078000	0.054561000	MigrationServer	Proxy	TLSv1	Application Data
29	1518	0.000013000	0.054574000	MigrationServer	Proxy	TCP	[TCP segment of a reassembled PDU]
30	1292	0.000009000	0.054583000	MigrationServer	Proxy	TLSv1	Application Data
31	64	0.000159000	0.054742000	Proxy	MigrationServer	TCP	9119[62830]62830 [ACK] Seq=1843722766 Ack=525499352 Win=2987 Len=0
32	484	0.000070000	0.054812000	MigrationServer	Proxy	TLSv1	Application Data, Application Data
33	64	0.000630000	0.055442000	Proxy	MigrationServer	TCP	9119[62830]62830 [ACK] Seq=1843722766 Ack=525500812 Win=2979 Len=0
34	64	0.000046000	0.055488000	Proxy	MigrationServer	TCP	9119[62830]62830 [ACK] Seq=1843722766 Ack=525503506 Win=2967 Len=0
35	64	0.000193000	0.055681000	Proxy	MigrationServer	TCP	9119[62830]62830 [ACK] Seq=1843722766 Ack=525503932 Win=2964 Len=0
36	196	0.368240000	0.423921000	Proxy	MigrationServer	TLSv1	Application Data, Application Data
37	292	0.045276000	0.469197000	Proxy	MigrationServer	TLSv1	Application Data, Application Data
38	64	0.000334000	0.469531000	MigrationServer	Proxy	TCP	62831[9119]9119 [ACK] Seq=653341697 Ack=3420581597 Win=511 Len=0
39	1518	0.003211000	0.472742000	MigrationServer	Proxy	TLSv1	Application Data
40	1518	0.000011000	0.472753000	MigrationServer	Proxy	TCP	[TCP segment of a reassembled PDU]
41	1292	0.000009000	0.472762000	MigrationServer	Proxy	TLSv1	Application Data
42	1518	0.000394000	0.473156000	MigrationServer	Proxy	TLSv1	Application Data
⋮							
67	64	0.000593000	0.476359000	Proxy	MigrationServer	TCP	9119[62830]62830 [ACK] Seq=1843722766 Ack=525529250 Win=2980 Len=0
68	356	0.380602000	0.856961000	Proxy	MigrationServer	TLSv1	Application Data, Application Data, Application Data
69	132	0.052061000	0.909022000	Proxy	MigrationServer	TLSv1	Application Data
70	64	0.000460000	0.909482000	MigrationServer	Proxy	TCP	62831[9119]9119 [ACK] Seq=653341697 Ack=3420581969 Win=510 Len=0
71	1518	0.006197000	0.915679000	MigrationServer	Proxy	TLSv1	Application Data

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)

```
48 64 0.000167000 0.473624000 Proxy MigrationServer TCP 9119[62830] [ACK] Seq=1843722766 Ack=525505392 Win=3072 Len=0
49 64 0.000050000 0.473674000 Proxy MigrationServer TCP 9119[62830] [ACK] Seq=1843722766 Ack=525508086 Win=3072 Len=0
50 64 0.000405000 0.474079000 Proxy MigrationServer TCP 9119[62830] [ACK] Seq=1843722766 Ack=525512240 Win=3060 Len=0
51 1518 0.000066000 0.474145000 MigrationServer Proxy TLSv1 Application Data
52 1518 0.000013000 0.474158000 MigrationServer Proxy TCP [TCP segment of a reassembled PDU]
53 1292 0.000009000 0.474167000 MigrationServer Proxy TLSv1 Application Data

#1 [3 Reassembled TCP Segments (4117 bytes): #51(1423), #52(1460), #53(1234)]
#2 [Secure Sockets Layer]
#3 [TLSv1 Record Layer: Application Data Protocol: Application Data]
Content Type: Application Data (23)
Version: TLS 1.0 (0x0301)
Length: 4112
Encrypted Application Data: 5faa5f1acf79c8a15959589de48179ce2593cc964ec5168e...
```

- 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

