Telescope

Peering Into the Depths of TLS Traffic in Real-Time

Caragea Radu

May 26, 2016

About me

- Bitdefender - Vulnerability Researcher
- PwnThyBytes: CTF team captain

This presentation

Is not about:

- Faults in the TLS protocol/implementation
- Attacks on the crypto level of TLS

This presentation

Is not about:

- Faults in the TLS protocol/implementation
- Attacks on the crypto level of TLS

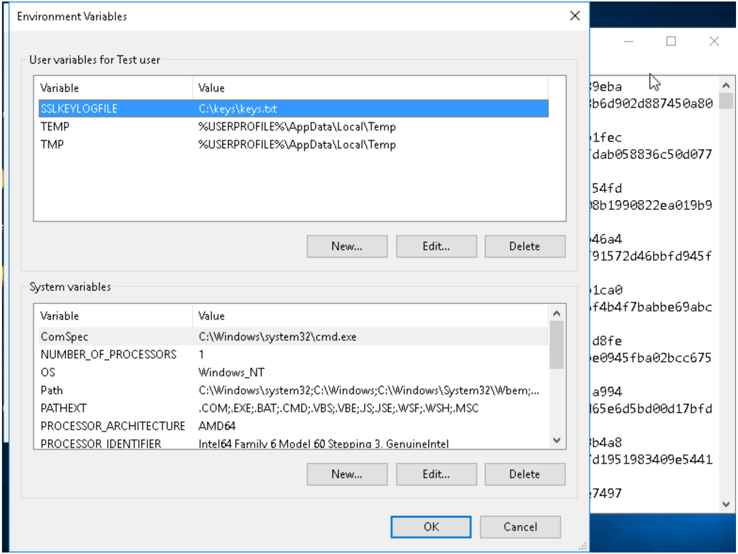
Is about:

- An experiment into what can be done from the hypervisor (powerful adversary; consider cloud providers)
- How far you can go and how noticeable it would be to an end-user
- Some unexpected results

Project Genesis

- Operating a honeypot farm
    - weak root (!) credentials: do what you will
    - analyze traffic (unless it's encrypted)
- Malvertising
    - Automated in-browser crawlers
    - Scour the net with the hope of getting infected
    - Need the infection vector (when served through TLS)

Other solutions: SSLKEYLOGFILE



User variables for Test user

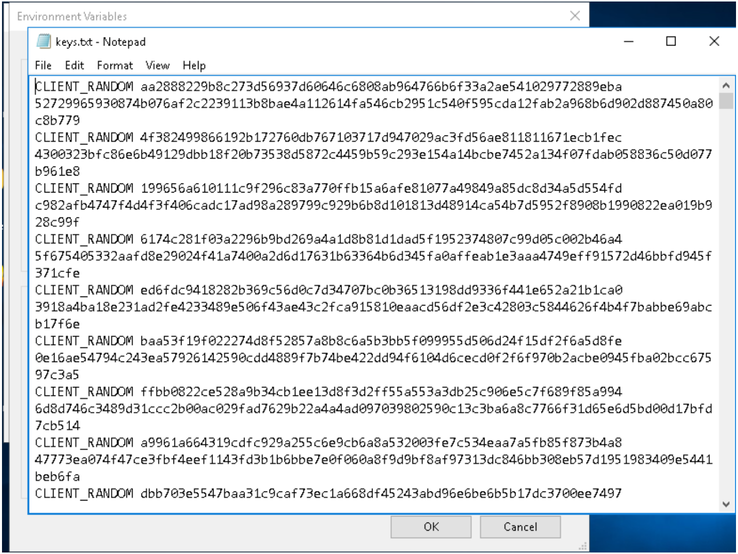
Variable	Value
SSLKEYLOGFILE	C:\keys\keys.txt
TEMP	%USERPROFILE%\AppData\Local\Temp
TMP	%USERPROFILE%\AppData\Local\Temp

System variables

Variable	Value
ComSpec	C:\Windows\system32\cmd.exe
NUMBER_OF_PROCESSORS	1
OS	Windows_NT
Path	C:\Windows\system32;C:\Windows;C:\Windows\System32\Wbem;...
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
PROCESSOR_ARCHITECTURE	AMD64
PROCESSOR_IDENTIFIER	Intel64 Family 6 Model 60 Stepping 3, GenuineIntel

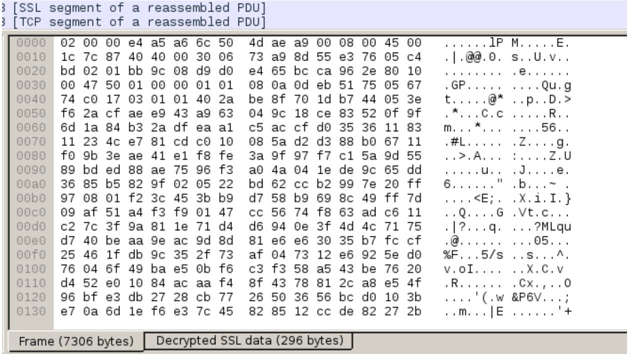
```
9eba
b6d902d887450a80

1fec
dab058836c50d077

54fd
8b1990822ea019b9

46a4
91572d46bbfd945f

1ca0
f4b4f7babbe69abc

d8fe
e0945fba02bcc675

a994
65e6d5bd00d17bfd

b4a8
d1951983409e5441

7497
```

Other solutions: SSLKEYLOGFILE



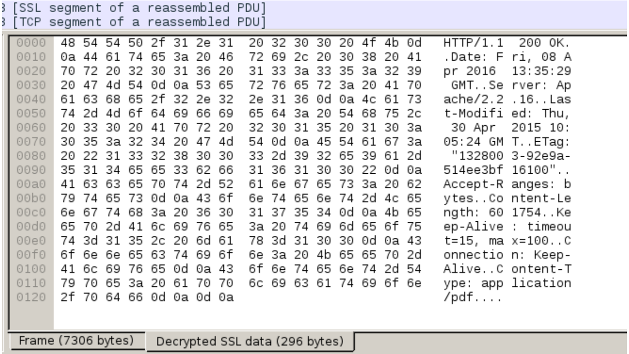
```
CLIENT_RANDOM aa2888229b8c273d56937d60646c6808ab964766b6f33a2ae541029772889eba
52729965930874b076af2c2239113b8bae4a112614fa546cb2951c540f595cda12fab2a968b6d902d887450a80
c8b779
CLIENT_RANDOM 4f382499866192b172760db767103717d947029ac3fd56ae811811671ecb1fec
4300323bfc86e6b49129dbb18f20b73538d5872c4459b59c293e154a14bcbe7452a134f07fdab058836c50d077
b961e8
CLIENT_RANDOM 199656a610111c9f296c83a770ffb15a6afe81077a49849a85dc8d34a5d554fd
c982afb4747f4d4f3f406cadc17ad98a289799c929b6b8d101813d48914ca54b7d5952f8908b1990822ea019b9
28c99f
CLIENT_RANDOM 6174c281f03a2296b9bd269a4a1d8b81d1dad5f1952374807c99d05c002b46a4
5f675405332aafd8e29024f41a7400a2d6d17631b63364b6d345fa0affeab1e3aaa4749eff91572d46bbfd945f
371cfe
CLIENT_RANDOM ed6fdc9418282b369c56d0c7d34707bc0b36513198dd9336f441e652a21b1ca0
3918a4ba18e231ad2fe4233489e506f43ae43c2fca915810eaacd56df2e3c42803c5844626f4b4f7babbe69abc
b17f6e
CLIENT_RANDOM baa53f19f022274d8f52857a8b8c6a5b3bb5f099955d506d24f15df2f6a5d8fe
0e16ae54794c243ea57926142590cdd4889f7b74be422dd94f6104d6cecd0f2f6f970b2acbe0945fba02bcc675
97c3a5
CLIENT_RANDOM ffbb0822ce528a9b34cb1ee13d8f3d2ff55a553a3db25c906e5c7f689f85a994
6d8d746c3489d31ccc2b00ac029fad7629b22a4a4ad097039802590c13c3ba6a8c7766f31d65e6d5bd00d17bfd
7cb514
CLIENT_RANDOM a9961a664319cdfc929a255c6e9cb6a8a532003fe7c534eaa7a5fb85f873b4a8
47773ea074f47ce3fbf4eef1143fd3b1b6bbe7e0f060a8f9d9bf8af97313dc846bb308eb57d1951983409e5441
beb6fa
CLIENT_RANDOM dbb703e5547baa31c9caf73ec1a668df45243abd96e6be6b5b17dc3700ee7497
```

Other solutions: SSLKEYLOGFILE



```
[SSL segment of a reassembled PDU]
[TCP segment of a reassembled PDU]
0000  02 00 00 e4 a5 a6 6c 50  4d ae a9 00 08 00 45 00   ......lP M.....E.
0010  1c 7c 87 40 40 00 30 06  73 a9 8d 55 e3 76 05 c4   .|.@@.0. s..U.v..
0020  bd 02 01 bb 9c 08 d9 d0  e4 65 bc ca 96 2e 80 10   ........ .e......
0030  00 47 50 01 00 00 01 01  08 0a 0d eb 51 75 05 67   .GP..... ....Qu.g
0040  74 c0 17 03 01 01 40 2a  be 8f 70 1d b7 44 05 3e   t.....@* ..p..D.>
0050  f6 2a cf ae e9 43 a9 63  04 9c 18 ce 83 52 0f 9f   .*...C.c .....R..
0060  6d 1a 84 b3 2a df ea a1  c5 ac cf d0 35 36 11 83   m...*... ....56..
0070  11 23 4c e7 81 cd c0 10  08 5a d2 d3 88 b0 67 11   .#L..... .Z....g.
0080  f0 9b 3e ae 41 e1 f8 fe  3a 9f 97 f7 c1 5a 9d 55   ..>.A... :....Z.U
0090  89 bd ed 88 ae 75 96 f3  a0 4a 04 1e de 9c 65 dd   .....u.. .J....e.
00a0  36 85 b5 82 9f 02 05 22  bd 62 cc b2 99 7e 20 ff   6......" .b...~ .
00b0  97 08 01 f2 3c 45 3b b9  d7 58 b9 69 8c 49 ff 7d   ....<E;. .X.i.I.}
00c0  09 af 51 a4 f3 f9 01 47  cc 56 74 f8 63 ad c6 11   ..Q....G .Vt.c...
00d0  c2 7c 3f 9a 81 1e 71 d4  d6 94 0e 3f 4d 4c 71 75   .|?...q. ...?MLqu
00e0  d7 40 be aa 9e ac 9d 8d  81 e6 e6 30 35 b7 fc cf   .@...... ...05...
00f0  25 46 1f db 9c 35 2f 73  af 04 73 12 e6 92 5e d0   %F...5/s ..s...^.
0100  76 04 6f 49 ba e5 0b f6  c3 f3 58 a5 43 be 76 20   v.oI.... ..X.C.v
0110  d4 52 e0 10 84 ac aa f4  8f 43 78 81 2c a8 e5 4f   .R...... .Cx.,..O
0120  96 bf e3 db 27 28 cb 77  26 50 36 56 bc d0 10 3b   ....'(.w &P6V...;
0130  e7 0a 6d 1e f6 e3 7c 45  82 85 12 cc de 82 27 2b   ..m...|E ......'+
```

Other solutions: SSLKEYLOGFILE

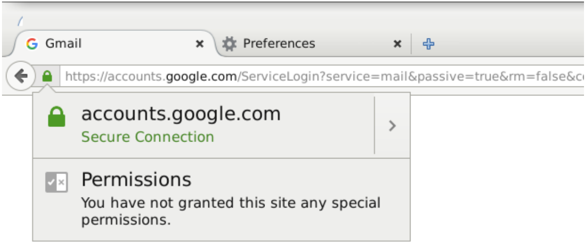


```
[SSL segment of a reassembled PDU]
[TCP segment of a reassembled PDU]
0000  48 54 54 50 2f 31 2e 31  20 32 30 30 20 4f 4b 0d   HTTP/1.1  200 OK.
0010  0a 44 61 74 65 3a 20 46  72 69 2c 20 30 38 20 41   .Date: F ri, 08 A
0020  70 72 20 32 30 31 36 20  31 33 3a 33 35 3a 32 39   pr 2016  13:35:29
0030  20 47 4d 54 0d 0a 53 65  72 76 65 72 3a 20 41 70    GMT..Se rver: Ap
0040  61 63 68 65 2f 32 2e 32  2e 31 36 0d 0a 4c 61 73   ache/2.2 .16..Las
0050  74 2d 4d 6f 64 69 66 69  65 64 3a 20 54 68 75 2c   t-Modifi ed: Thu,
0060  20 33 30 20 41 70 72 20  32 30 31 35 20 31 30 3a    30 Apr  2015 10:
0070  30 35 3a 32 34 20 47 4d  54 0d 0a 45 54 61 67 3a   05:24 GM T..ETag:
0080  20 22 31 33 32 38 30 30  33 2d 39 32 65 39 61 2d    "132800 3-92e9a-
0090  35 31 34 65 65 33 62 66  31 36 31 30 30 22 0d 0a   514ee3bf 16100"..
00a0  41 63 63 65 70 74 2d 52  61 6e 67 65 73 3a 20 62   Accept-R anges: b
00b0  79 74 65 73 0d 0a 43 6f  6e 74 65 6e 74 2d 4c 65   ytes..Co ntent-Le
00c0  6e 67 74 68 3a 20 36 30  31 37 35 34 0d 0a 4b 65   ngth: 60 1754..Ke
00d0  65 70 2d 41 6c 69 76 65  3a 20 74 69 6d 65 6f 75   ep-Alive : timeou
00e0  74 3d 31 35 2c 20 6d 61  78 3d 31 30 30 0d 0a 43   t=15, ma x=100..C
00f0  6f 6e 6e 65 63 74 69 6f  6e 3a 20 4b 65 65 70 2d   onnectio n: Keep-
0100  41 6c 69 76 65 0d 0a 43  6f 6e 74 65 6e 74 2d 54   Alive..C ontent-T
0110  79 70 65 3a 20 61 70 70  6c 69 63 61 74 69 6f 6e   ype: app lication
0120  2f 70 64 66 0d 0a 0d 0a                            /pdf....
```

Other solutions: SSLKEYLOGFILE

- Implemented in libNSS (firefox) and openSSL (chrome) but not IE/Edge
- The downside: it's blatantly visible

Other solutions: Custom Root CA



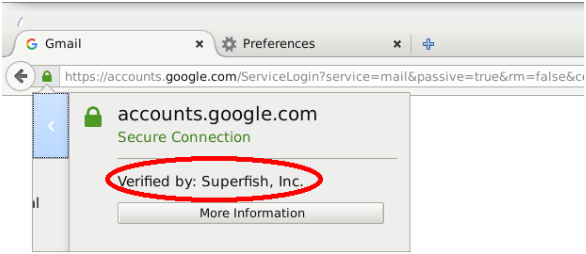
accounts.google.com

Secure Connection

Permissions

You have not granted this site any special permissions.

Other solutions: Custom Root CA



https://accounts.google.com/ServiceLogin?service=mail&passive=true&rm=false&c

accounts.google.com

Secure Connection

Verified by: Superfish, Inc.

Other solutions: Custom Root CA

- Typical solution used by AVs/proxies to intercept TLS traffic
- Visible by malware by scanning the disk.
- Moreover, in "Analyzing Forged SSL Certificates in the Wild" Huang et al show how to do this within the browser

Other solutions: PANDA keyfind plugin

Finding SSL/TLS Master Secrets with PANDA

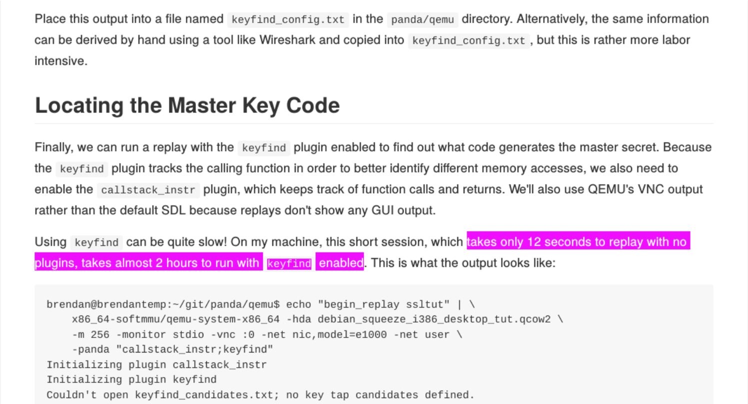
Introduction

Monitoring SSL/TLS-encrypted traffic is a classic problem for intrusion detection systems. Currently, hypervisor- or network-based IDSes that wish to analyze encrypted traffic must perform a man-in-the-middle attack on the connection, presenting a false server certificate to the client. Not only does this require the client to cooperate by trusting certificates signed by the intrusion detection system, it also takes control of the certificate verification process out of the hands of the client---a dangerous step, given that many existing SSL/TLS interception proxies have a history of certificate trust vulnerabilities.

Instead of a man-in-the-middle attack, we can instead attempt to locate the code that generates SSL/TLS master secret; this secret is sufficient to decrypt any encrypted traffic in a given session, giving us a "man-on-the-inside". Once we have identified the location of the code that generates this secret, we can hook it using any number of standard techniques in order to dump out the master secret. This secret can then be provided to an IDS to decrypt the content of the SSL stream; it may also be provided to a tool like Wireshark to decrypt packet captures after the fact (even if perfect forward secrecy is used).

- really cool solution!
- run the machine under QEMU
- use pre-established "trace points"
- trace memory writes

Other solutions: PANDA keyfind plugin



Place this output into a file named `keyfind_config.txt` in the `panda/qemu` directory. Alternatively, the same information can be derived by hand using a tool like Wireshark and copied into `keyfind_config.txt`, but this is rather more labor intensive.

Locating the Master Key Code

Finally, we can run a replay with the `keyfind` plugin enabled to find out what code generates the master secret. Because the `keyfind` plugin tracks the calling function in order to better identify different memory accesses, we also need to enable the `callstack_instr` plugin, which keeps track of function calls and returns. We'll also use QEMU's VNC output rather than the default SDL because replays don't show any GUI output.

Using `keyfind` can be quite slow! On my machine, this short session, which takes only 12 seconds to replay with no plugins, takes almost 2 hours to run with `keyfind` enabled. This is what the output looks like:

```
brendan@brendantemp:~/git/panda/qemu$ echo "begin_replay ssltut" | \
    x86_64-softmmu/qemu-system-x86_64 -hda debian_squeeze_i386_desktop_tut.qcow2 \
    -m 256 -monitor stdio -vnc :0 -net nic,model=e1000 -net user \
    -panda "callstack_instr;keyfind"
Initializing plugin callstack_instr
Initializing plugin keyfind
Couldn't open keyfind_candidates.txt; no key tap candidates defined.
```

- huge overhead
- non portable

Scavenging the keys from memory?

While connection is still active, the keys must still be in memory.

Scavenging the keys from memory?

While connection is still active, the keys must still be in memory.

- Problem 1: exact key location is unknown
    - need to dump all memory
    - dumping memory takes time ($>$10 seconds for 4 GB)
    - multiple connections occur one after the other or interspersed
    - space quickly fills up

Scavenging the keys from memory?

While connection is still active, the keys must still be in memory.

- Problem 1: exact key location is unknown
    - need to dump all memory
    - dumping memory takes time ($>$10 seconds for 4 GB)
    - multiple connections occur one after the other or interspersed
    - space quickly fills up
- Problem 2: we don't even know how to distinguish the correct keys from random memory.

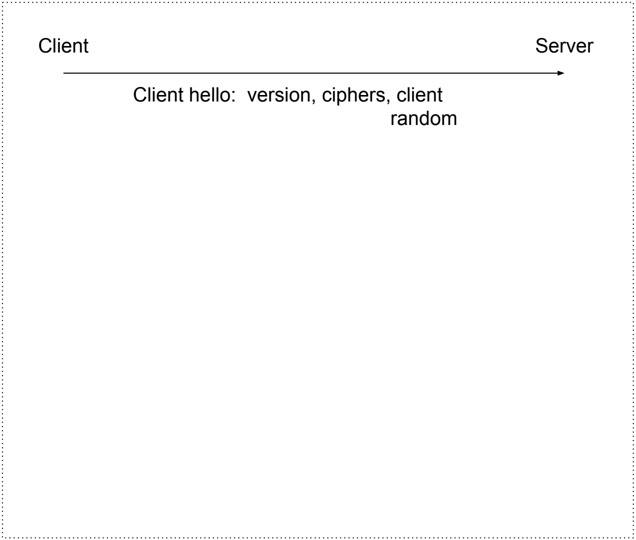
Scavenging the keys from memory?

While connection is still active, the keys must still be in memory.

- Problem 1: exact key location is unknown
  - need to dump all memory
  - dumping memory takes time ($>$10 seconds for 4 GB)
  - multiple connections occur one after the other or interspersed
  - space quickly fills up
- Problem 2: we don't even know how to distinguish the correct keys from random memory.

To understand our approach we must dig deeper!

How exactly does TLS work? Client Hello

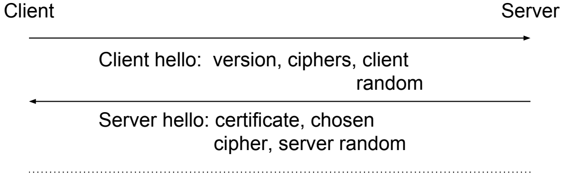


Client

Server

Client hello: version, ciphers, client random

How exactly does TLS work? Server Hello



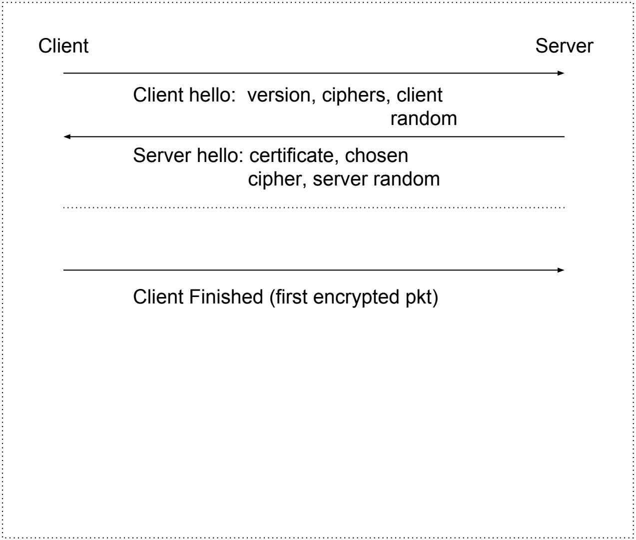
Client

Server

Client hello: version, ciphers, client random

Server hello: certificate, chosen cipher, server random

How exactly does TLS work? Client Finished



Client

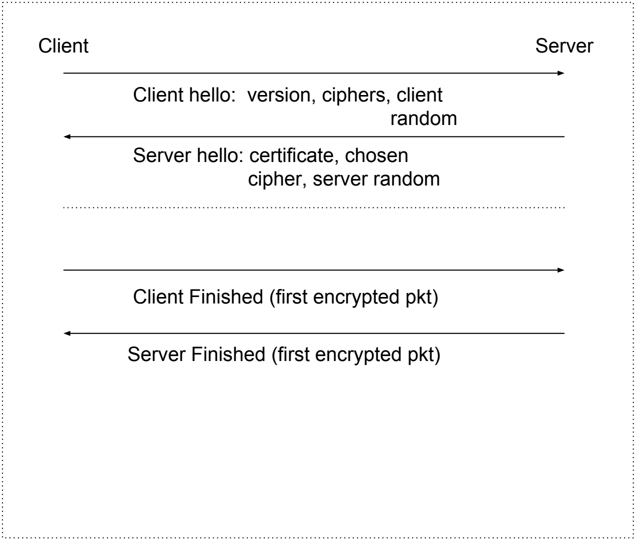
Server

Client hello: version, ciphers, client random

Server hello: certificate, chosen cipher, server random

Client Finished (first encrypted pkt)

How exactly does TLS work? Server Finished



Client

Server

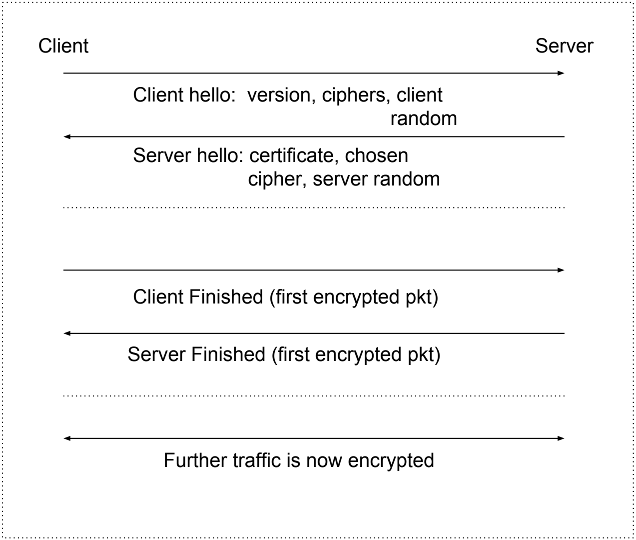
Client hello: version, ciphers, client random

Server hello: certificate, chosen cipher, server random

Client Finished (first encrypted pkt)

Server Finished (first encrypted pkt)

How exactly does TLS work? Handshake Complete



Client

Server

Client hello: version, ciphers, client random

Server hello: certificate, chosen cipher, server random

Client Finished (first encrypted pkt)

Server Finished (first encrypted pkt)

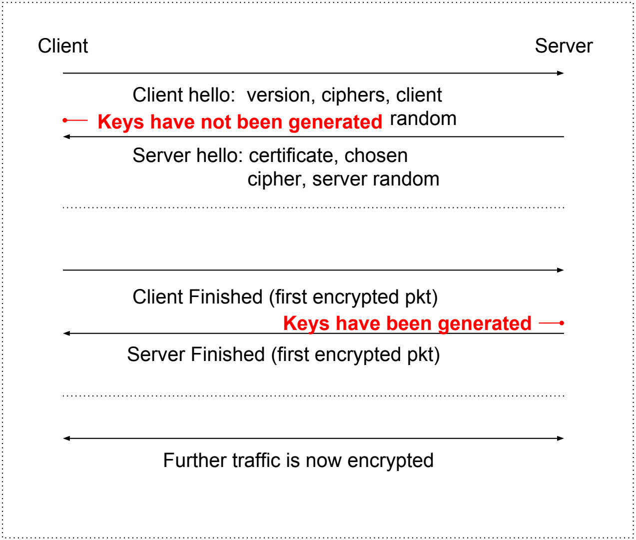
Further traffic is now encrypted

But wait!

Excerpt from RFC5246/4346:

```
To generate the key material, compute
key_block = PRF(SecurityParameters.master_secret,
                "key expansion",
                SecurityParameters.server_random +
                SecurityParameters.client_random);

until enough output has been generated.
Key material = [client_write_MAC_key][server_write_MAC_key]
               [client_write_key][server_write_key]
               [client_write_IV][server_write_IV]
```

Key events in the TLS handshake



Client

Server

Client hello: version, ciphers, client random

Keys have not been generated

Server hello: certificate, chosen cipher, server random

Client Finished (first encrypted pkt)

Keys have been generated

Server Finished (first encrypted pkt)

Further traffic is now encrypted

Implications

- Only track memory between events
- Events signalled by passing through netfilter queue
- Dramatic decrease in memdump size

Implications

- Only track memory between events
- Events signalled by passing through netfilter queue
- Dramatic decrease in memdump size
- But how do you actually "track" pages?

Think VM live migration

Think VM live migration

Logdirty mechanism

- t_0: start tracking pages written to from t_0 and flush the RAM to the target on the network

Think VM live migration

Logdirty mechanism

- t_0: start tracking pages written to from t_0 and flush the RAM to the target on the network
- when this finishes, get the "dirty" pages (at t_1) and send the delta to target again

Think VM live migration

Logdirty mechanism

- t_0: start tracking pages written to from t_0 and flush the RAM to the target on the network
- when this finishes, get the "dirty" pages (at t_1) and send the delta to target again
- repeat this for every t_i - t_{i+1} until number of pages is under threshold

Think VM live migration

Logdirty mechanism

- t_0: start tracking pages written to from t_0 and flush the RAM to the target on the network
- when this finishes, get the "dirty" pages (at t_1) and send the delta to target again
- repeat this for every t_i - t_{i+1} until number of pages is under threshold
- stop VM1, do iteration one last time, start VM2

Logdirty

Similar mechanisms exist in most (all) modern hypervisors that support VM migration

- Page fault based (basic)
- EPT A/D

Logdirty

Similar mechanisms exist in most (all) modern hypervisors that support VM migration

- Page fault based (basic)
- EPT A/D
- Recently, a processor extension especially for this: Intel PML. Convenient, right ???

Putting the TeLeScope together

- Filter target network events and send to netfilter queue
- Start logging on Server Hello
- Stop logging and dump pages on Client Finished
- The result is a micro-memdump
- Can be processed **offline** anytime

TeLeScope results

- on Linux VM per connection: 500K - 10 MB memdump
- on Windows VM per connection: 15 MB - 60 MB memdump
- VM pause time: under 0.5 ms but on average 0.05 ms
- page dump time: 1-10 ms (disguised as packet delay)

Problems revisited

- Problem 1: you don't know where the keys are **(partially solved)**
    - ~~need to dump all memory~~
    - ~~dumping memory takes time ($>$10 seconds for 4 GB)~~
    - multiple connections occur one after the other or interspersed
    - space ~~quickly~~ fills up
- Problem 2: we don't even know how to distinguish the correct keys from random memory.

Problem 2: key discerning

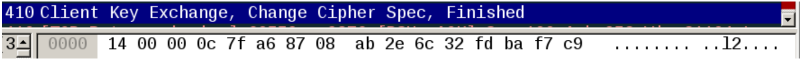
Apparently multiple unknowns

- key format
- key parameters: IV, nonce, etc
- what to encrypt/decrypt
- what it decrypts to

Known Plaintext Attack

The Client/Server Finished messages have a fixed form:

`14 00 00 0C [12 random bytes]`

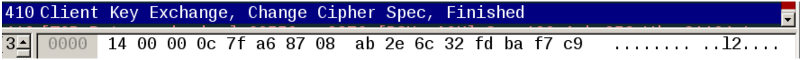


```
410 Client Key Exchange, Change Cipher Spec, Finished
0000  14 00 00 0c 7f a6 87 08  ab 2e 6c 32 fd ba f7 c9   ........ ..l2....
```

Known Plaintext Attack

The Client/Server Finished messages have a fixed form:

`14 00 00 0C [12 random bytes]`



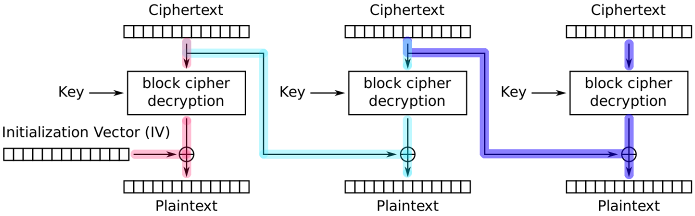
```
410 Client Key Exchange, Change Cipher Spec, Finished
3  0000  14 00 00 0c 7f a6 87 08  ab 2e 6c 32 fd ba f7 c9   ........ ..l2....
```

- $\frac{1}{2^{32}}$ chance of a False Positive
- This works if you can decrypt the first 4 bytes (think stream ciphers, AES/CTR, etc)

Alexa top 1000 ciphers

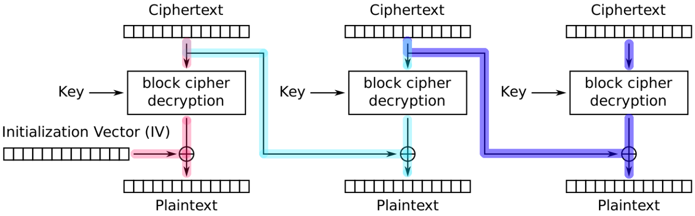
- 5% RC4
- 21% AES CBC
- 73% AES GCM

AES CBC



- Decrypting each block depends on having the previous block
- For the first block you need the IV (not explicit for TLS 1.0)
- The known plaintext is exactly in the first block

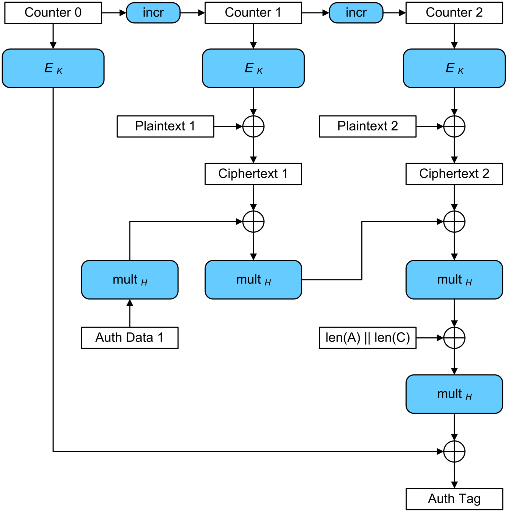
AES CBC



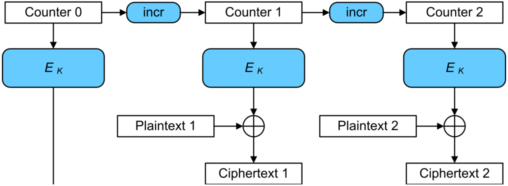
- Decrypting each block depends on having the previous block
- For the first block you need the IV (not explicit for TLS 1.0)
- The known plaintext is exactly in the first block
- We use the last block for the padding

```
0000   14 00 00 0c 7f a6 87 08 ab 2e 6c 32 fd ba f7 c9   ..........l2....
0010   9c c5 76 20 da 83 6e b5 27 af ac ce e0 ac 1a e4   ..v ..n.'.......
0020   8c 86 55 fc 0b 0b 0b 0b 0b 0b 0b 0b 0b 0b 0b 0b   ..U.............
```

AES GCM/CTR

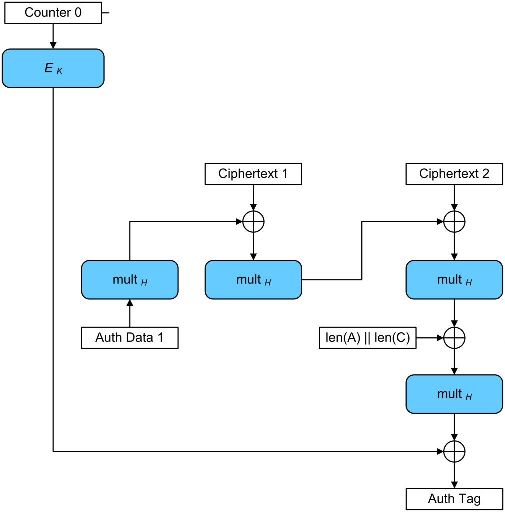

Counter 0
incr
Counter 1
incr
Counter 2
E_K
E_K
E_K
Plaintext 1
Plaintext 2
Ciphertext 1
Ciphertext 2
mult $_H$
mult $_H$
mult $_H$
Auth Data 1
len(A) || len(C)
mult $_H$
Auth Tag

AES GCM/CTR Encryption



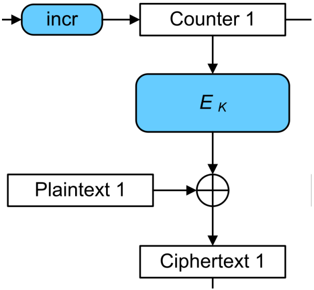
AES CTR

AES GCM/CTR Authentication



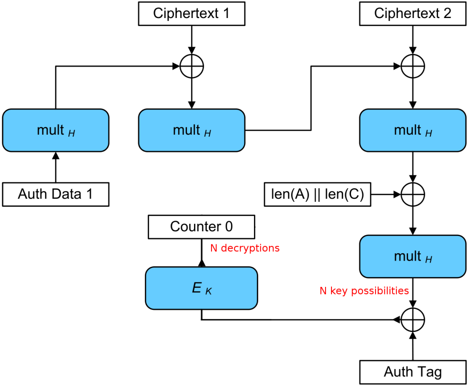
AES GCM

AES GCM bruteforce attempt



- Counter is [8 bytes key material][8 bytes counter]
- Here: 0fddf45e89838e700000000000000001
- The first half is also from the key material
- Implies $O(N^2)$ which we don't like!

AES GCM tag



- Auth Data, the Ciphertexts, Lengths and the Tag Known
- Each key K corresponds to one H
- Reverse the Flow and do one extra decryption
- Known Plaintext Attack on Counter format

Telesync benchmarks

- 1 thread, 240 MB completely random data: 6750 ms
- 6 threads, 240 MB completely random data: 557 ms (12x speedup)
- 6 threads + heuristics, 240 MB typical memdump data: 151 ms (44x speedup)

Telesync benchmarks

- 1 thread, 240 MB completely random data: 6750 ms
- 6 threads, 240 MB completely random data: 557 ms (12x speedup)
- 6 threads + heuristics, 240 MB typical memdump data: 151 ms (44x speedup)

However, a typical memdump size is usually 5-10 times smaller. So we can usually consume as fast as we can produce!

Demo 1 (manual)

Demo 2 (integrated)

Technique genericity

Actually, this can be applied to other protocols that use a similar negotiation technique for the symmetric keys:

- VPN
- SSH
- Tor

Conclusions and more

- Decrypting TLS on current implementations is definitely feasible with a hypervisor-in-the-middle attack
- We developed a fast and efficient PoC
- * You might not observe if you are the one ”under scrutiny” on a VPS
- * Actually, if you’re not in control of the bare metal all bets are off

Questions

?