TLS Security – Where Do We Stand?

Kenny Paterson

(based on joint work with Nadhem AlFardan, Dan Bernstein, Bertram Poettering, Jacob Schuldt)
Outline

• TLS and the TLS Record Protocol
• Lucky 13 attack
• RC4 attack
• Discussion
TLS

• SSL = Secure Sockets Layer.
  – Developed by Netscape in mid 1990s.
  – SSLv3 still widely supported.

• TLS = Transport Layer Security.
  – IETF-standardised version of SSL.
  – TLS 1.0 in RFC 2246 (1999).

• SSL/TLS has become the *de facto* secure protocol of choice for the web.
TLS Protocol Architecture

- Handshake Protocol
- Change Cipher Spec Protocol
- Alert Protocol
- HTTP, other apps

Record Protocol

TCP
TLS Record Protocol: MAC-Encode-Encrypt

MAC

SQN || HDR

Payload

MAC

Payload

MAC tag

Padding

Encrypt

HDR

Ciphertext

HMAC-MD5, HMAC-SHA1, HMAC-SHA256

CBC-AES128, CBC-AES256, CBC-3DES, RC4-128

“00” or “01 01” or “02 02 02” or …. or “FF FF ….FF”
AE and TLS

• Authenticated Encryption (AE) also available in TLS 1.2.

• AES-GCM in RFC 5288.
• AES-CCM in RFC 6655.

• But TLS 1.2 is not yet widely supported.
  – Most browsers support SSLv3 and TLS 1.0 only.
  – Currently, roughly 50/50 usage split between CBC-mode and RC4.
  – Less than 15% of servers support TLS 1.1 or higher
    (source: SSL Pulse).
TLS Needs Uniform Errors

• TLS-CBC has had a troubled adolescence.
  – Padding oracles, BEAST.

• Practical padding oracle attacks against TLS-CBC if the attacker can distinguish padding failures from MAC failures during decryption.

• For this reason, a uniform error assumption is needed in formal security analysis of TLS-CBC.

• So how should TLS implementations ensure uniform errors?
Ensuring Uniform Errors

From the TLS 1.2 specification:

…the implementations MUST ensure that record processing time is essentially the same whether or not the padding is correct.

In general, the best way to do this is to compute the MAC even if the padding is incorrect, and only then reject the packet.

Compute the MAC on what though?
Ensuring Uniform Errors

*For instance, if the pad appears to be incorrect, the implementation might assume a zero-length pad and then compute the MAC.*

• This approach is adopted in many implementations, including OpenSSL, NSS (Chrome, Firefox), BouncyCastle, OpenJDK, …

• One alternative (GnuTLS and others) is to remove as many bytes as are indicated by the last byte of plaintext and compute the MAC on what’s left.
Outline

• TLS and the TLS Record Protocol
• Lucky 13 attack
• RC4 attack
• Discussion
Lucky 13

- Distinguishing attacks and full plaintext recovery attacks against TLS-CBC implementations following the advice in the RFC.
  - And variant attacks against those that do not.

- Applies to all versions of SSL/TLS.
  - SSLv3.0, TLS 1.0, 1.1, 1.2.
  - And DTLS.

- Demonstrated in the lab against OpenSSL and GnuTLS.
Reminder: MAC-Encode-Encrypt in TLS

SQN || HDR → Payload → MAC → Payload → MAC tag → Padding → Encrypt → HDR → Ciphertext
Lucky 13 – Basic Idea

• TLS decryption removes padding and MAC tag to get PAYLOAD.

• HMAC computed on SQN || HDR || PAYLOAD.

• HMAC computation involves iteration of hash compression function, e.g. MD5, SHA-1, SHA-256.

• Running time of HMAC depends on $L$, the byte length of SQN || HDR || PAYLOAD:
  – $L \leq 55$ bytes: 4 compression functions;
  – $56 \leq L \leq 119$: 5 compression functions;
  – $120 \leq L \leq 183$: 6 compression functions;
  – ....
Lucky 13 Distinguishing Attack

\[ C = \text{Encrypt}(K, M) \]

M is either \( R^{287} || 00 \) or \( R^{32} || FF^{256} \)

- Adversary intercepts \( C \), mauls, and forwards on to recipient.
- Time taken to respond with error message will indicate whether \( M = R^{287} || 00 \) or \( R^{32} || FF^{256} \).
- Ciphertext-only distinguishing attack.
Lucky 13 Distinguishing Attack – Choose

\[
\begin{array}{c}
\text{R} \\
\text{R R} \\
\text{R 00}
\end{array}
\]

\[
\begin{array}{c}
\text{IV} \\
\text{C}
\end{array}
\]

\[
\begin{array}{c}
\text{R} \\
\text{FF FF}
\end{array}
\]

\[
\begin{array}{c}
\text{IV} \\
\text{C}
\end{array}
\]
Lucky 13 Distinguishing Attack – Maul
Lucky 13 Distinguishing Attack – Inject

- 1-byte valid padding
- 20-byte MAC
- 267-byte message

- 256-byte valid padding
- 20-byte MAC
- 12-byte message
Lucky 13 Distinguishing Attack – Decrypt
Lucky 13 Distinguishing Attack – Decrypt

Timing difference: 4 SHA-1 compression function evaluations
Experimental Results for Distinguishing Attack

- OpenSSLv1.0.1 on server running at 1.87Ghz.
- 100 Mbit LAN.
- Difference in means is circa 3.2 μs.
## Success Probability

<table>
<thead>
<tr>
<th>Number of Sessions</th>
<th>Success Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.756</td>
</tr>
<tr>
<td>4</td>
<td>0.858</td>
</tr>
<tr>
<td>16</td>
<td>0.951</td>
</tr>
<tr>
<td>64</td>
<td>0.992</td>
</tr>
<tr>
<td>128</td>
<td>1</td>
</tr>
</tbody>
</table>
Lucky 13 – Plaintext Recovery

(HMAC-SHA-1 + AES-CBC)

(XOR 2-byte Δ here and submit for decryption)

Produces valid patterns “01 01” or “00”, OR bad pad.
Case: “01 01” (or longer valid pad)

\[ IV \rightarrow R_1 \rightarrow R_2 \rightarrow C_{t-1} \rightarrow C_t \]

\( d_K \rightarrow \oplus \rightarrow d_K \rightarrow \oplus \rightarrow d_K \rightarrow \oplus \rightarrow d_K \)

XOR 2-byte \( \Delta \) here and submit for decryption

\( \text{SQN} || \text{HDR} \)

13 + 16 + 16 + 10 = 55 bytes

20 bytes

“01 01”

4 SHA-1 compression function evaluations
Case: “00”

XOR 2-byte \( \Delta \) here and submit for decryption

\[
IV \xrightarrow{d_K} R_1 \xrightarrow{d_K} R_2 \xrightarrow{d_K} C_{t-1} \xrightarrow{d_K} C_t
\]

SQN||HDR

56 bytes

20 bytes

“00”

5 SHA-1 compression function evaluations
Case: Bad padding

XOR 2-byte Δ here and submit for decryption

$IV \rightarrow R_1 \rightarrow R_2 \rightarrow C_{t-1} \rightarrow C_t$

$\text{SQN}\|\text{HDR}$

57 bytes

20 bytes

zero-length pad

5 SHA-1 compression function evaluations
Lucky 13 – Plaintext Recovery

• The injected ciphertext causes bad padding and/or a bad MAC.
  – This leads to a TLS error message, which the attacker times.

• There is a timing difference between “01 01” case and the other 2 cases.
  – A single SHA-1 compression function evaluation.
  – Roughly 1000 clock cycles, 1μs range on typical processor.
  – Measurable difference on same host, LAN, or a few hops away.

• Detecting the “01 01” case allows last 2 plaintext bytes in the target block $C_t$ to be recovered.
  – Some standard CBC algebra.
  – Attack then extends easily to all bytes.
Lucky 13 – Attack Cost

- We need $2^{16}$ attempts to try all 2-byte $\Delta$ values.

- And we need around $2^7$ trials per $\Delta$ value to reliably distinguish the different events.
  - Noise level depends on experimental set-up.

- Each trial kills the TLS session.

- Hence the headline attack cost is $2^{23}$ sessions, all encrypting the same plaintext.

- So what’s all the fuss about?
Lucky 13 – Improvements

• If 1-out-of-2 last bytes known, then we only need $2^8$ attempts per byte.

• If the plaintext is base64 encoded, then we only need $2^6$ attempts per byte.
  – And $2^7$ trials per attempt to de-noise, for a total of $2^{13}$.

• BEAST-style attack targeting HTTP cookies.
  – Malicious client-side Javascript makes HTTP GET requests.
  – TLS sessions are automatically generated and HTTP cookies attached.
  – Pad GET requests so that 1-out-of-2 condition always holds.
  – Cost of attack is $2^{13}$ GET requests per byte of cookie.
  – Now a practical attack!
Experimental Results

- Byte 14 of plaintext set to 01; byte 15 set to FF.
- OpenSSLv1.0.1 on server running at 1.87Ghz.
- 100 Mbit LAN.
- Median times (noise not shown).
OpenSSL: recovering last byte in a block, using percentile test to extract correct byte value, no assumptions on plaintext.
Lucky 13 – Further Extensions

• The attack extends to other MAC algorithms.
  – Interplay between block-size, MAC tag size and 13-byte field SQN || HDR.

• The attack extends to other methods for dealing with bad padding.
  – e.g. as in GnuTLS, faster but partial plaintext recovery.

• The attack can be applied to DTLS.
  – No error messages, but simulate these via DTLS Heartbeats.
  – Errors non-fatal, so can execute attack in a single session.
  – Cam amplify timing differences using AlFardan-Paterson techniques (NDSS 2012).
Lucky 13 – Impact

- **OpenSSL** patched in versions 1.0.1d, 1.0.0k and 0.9.8y, released 05/02/2013.
- **NSS** (Firefox, Chrome) patched in version 3.14.3, released 15/02/2013.
- **Opera** patched in version 12.13, released 30/01/2013
- **Oracle** released a special critical patch update of JavaSE, 19/02/2012.
- **BouncyCastle** patched in version 1.48, 10/02/2013
- Also **GnuTLS, PolarSSL, CyaSSL, MatrixSSL**.
- **Microsoft** “determined that the issue had been adequately addressed in previous modifications to their TLS and DTLS implementation”.
- **Apple**: status unknown.

- Full details at: www.isg.rhul.ac.uk/tls/Lucky13.html
Lucky 13 – Countermeasures

- We need constant time decryption for TLS-CBC.

- Add dummy hash compression function computations when padding is *good* to ensure total is the same as when padding is *bad*.

- Add dummy padding checks to ensure number of iterations done is independent of padding length and/or correctness of padding.

- Watch out for length sanity checks too.
  - Need to ensure there’s enough space for *some* plaintext after removing pad and MAC, but without leaking any information about amount of padding removed.
Performance of Countermeasures

Before

After
Other Countermeasures?

- Introduce random delays during decryption.
  - Surprisingly *ineffective*, analysis in our paper.

- Use TLS’s RC4 stream cipher option instead of CBC-mode.
  - No padding to worry about.
  - Keep listening to find out why not!

- Redesign TLS:
  - Pad-MAC-Encrypt or Pad-Encrypt-MAC.
  - No easy deployment route, seems unlikely to happen.

- Switch to TLS 1.2
  - Has support for AES-GCM and AES-CCM.
Outline

• TLS and the TLS Record Protocol
• Lucky 13 attack
• RC4 attack
• Discussion
RC4 Attack

• The RC4 stream cipher has long been known to have statistical weaknesses.
  - e.g. Mantin-Shamir bias, recent work of Maitra et al., Isobe et al.
  - Most attention has been given to the initial few bytes of keystream.
  - Academic focus has been on finding and giving theoretical explanations for individual biases.

• Usual countermeasure is to discard the initial bytes of keystream and use only “good” bytes.

• So what does RC4 in TLS do?
RC4 Attack

- TLS does not discard bytes!

- Why not?
  - Because it hurts performance too much;
  - The biases are small anyway; and only exist in the first few bytes.

- We estimated the biases in the first 256 output bytes by sampling RC4 keystreams for $2^{44}$ random 128-bit keys.

- We found many previously unreported biases of significant size…
  - [www.isg.rhul.ac.uk/tls/biases.pdf](http://www.isg.rhul.ac.uk/tls/biases.pdf)
Biases in byte 16 of RC4 Output

Ciphertext distribution at position 16

Probability

Byte value [0...255]
Biases in byte 31 of RC4 Output
Biases in byte 128 of RC4 Output

Ciphertext distribution at position 128

Probability

Byte value [0...255]
RC4 Attack

- These biases are large enough to enable plaintext recovery attacks on first 256 bytes of TLS session.

- Needs a multi-session attack, using BEAST-style malware.

- Attack using simple Bayesian technique:
  - For each position $i$:
    $$ C_i = P_i \oplus K_i $$
  - Many samples of $C_i$, guess $P_i$, induces distribution on $K_i$.
  - Estimate likelihood of induced distribution using measured distribution on $K_i$.
  - Select as correct plaintext byte the candidate $P_i$ giving highest likelihood.
Success probability: $2^{24}$ sessions
Success probability: $2^{26}$ sessions
Success probability: $2^{28}$ sessions
Success probability: $2^{30}$ sessions
Success probability: $2^{32}$ sessions
Possible Attack Enhancements

• Many sessions are needed.

• The attack can only target the first 256 output bytes.
  – Containing less interesting HTTP headers.

• We solve both problems using 2-byte Fluhrer-McGrew biases.
  – Smaller biases, but persistent throughout keystream.
  – Arrange for HTTP session cookie to be repeatedly sent at predictable locations in keystream.
  – Use Viterbi-style algorithm to do ML estimation of plaintext bytes.
  – Roughly $2^{35}$ encryptions needed for reliable recovery.
Results for 2-byte Biases

x-axis: units of $2^{30}$ encryptions.
Blue line: success rate for 16-byte plaintext recovery.
Red line: success rate for individual byte recovery.
RC4 Attack – Countermeasures

• Can’t just discard initial output bytes without updating all clients and servers simultaneously.
  – And doesn’t help against 2-byte attacks anyway.

• Discussions with vendors on ad hoc measures for HTTP.
  – Randomisation.
  – Burn-off initial bytes via short messages.
  – Put limits on number of times cookies can be sent.
RC4 Attack – Impact

• Fewer vendors have reacted publicly.
  – Google focussed on implementing TLS 1.2.
  – Opera has implemented cookie limit countermeasure.

• Lots of press coverage.

• We can expect the amount of RC4 traffic to drop in the coming months.

• Further details at: www.isg.rhul.ac.uk/tls
Outline

• TLS and the TLS Record Protocol
• Lucky 13 attack
• RC4 attack
• Discussion
Discussion

• MAC-Encode-Encrypt is hard to implement securely and hard to prove positive security results about.
  – Long history of attacks and fixes.
  – Actual TLS-CBC construction was only fully analysed in 2011.
  – 500 line patch to OpenSSL to stop Lucky 13 attack.

• RC4 was known to be weak for many years.
  – Actual exploitation of weaknesses in a TLS context went unexplored.
  – Needed multi-session mechanism (BEAST technology) to make it plausible.
    • Traditional cryptanalysis insufficient.
Discussion

• Once a bad cryptographic choice is out there in implementations, it’s very hard to “recall”.
  – Old versions of TLS hang around for a long time.
  – Slow uptake of TLS 1.2.

• TLS is coming under sustained pressure from attacks.
  – BEAST, Lucky 13 and RC4 attacks are providing incentives to move to TLS 1.2.
  – Attacks are “semi-practical” but we ignore such attacks at our peril.
  – Good vendor response to Lucky 13, less so to RC4 attack.
    • One is fixable, the other not (really).
Further Reading


• [www.isg.rhul.ac.uk/tls](http://www.isg.rhul.ac.uk/tls)