Bar Mitzvah Attack: Breaking SSL with 13-Year Old RC4 Weakness

Itsik Mantin, Imperva
Why Bar Mitzvah?

- בר מיזווה (Hebrew)
- According to Jewish tradition, when Jewish boys become 13 years old, they become accountable for their actions and become a **Bar Mitzvah**.
- The attack is based on a vulnerability in RC4 that was “born” (discovered) 13 years ago and recently (August 2014) “celebrated” it’s Bar-Mitzvah.
- The Invariance Weakness
On TLS

- On TLS
- On RC4
- The Invariance Weakness
- The Attacks
- Conclusion
From SSL to TLS

• The Secure Socket Layer
  — Developed by Netscape for https communication
  — SSL 3.0 (RFC 6101) released in 1996

• Renamed to Transport Layer Security in 1999
  — TLS 1.0 (RFC 2246, 1999)
  — TLS 1.1 (RFC 4346, 2006)
  — TLS 1.2 (RFC 5246, 2008)
  — TLS1.3: work in progress
TLS Protocol Support

- SSL-Pulse (March 9, 2015)
TLS Objectives

- Mutual Authentication
  - Usually only Server authentication is used
- Data Protection
  - Data Integrity
  - Data Confidentiality
Passive Attacker (Sniffing)

alice.wonder@gmail.com
Alice123!
Man-in-the-Middle Attacker (MitM)

alice.wonder@gmail.com
Alice123!

alice.wonder@gmail.com
Alice123!
TLS Security

- Cipher attacks (BEAST, RC4 (Royal Holloway))
- Compression attacks (CRIME, TIME, BREACH)
- Downgrade attacks (POODLE)
- Padding Oracle attacks (Lucky13)
- Implementation attacks (Heartbleed)
On RC4

- On TLS
- **On RC4**
- The Invariance Weakness
- The Attacks
- Conclusion
RC4 Usage in TLS

- SSL-Pulse (March 9, 2015)

- Not supported: 37,840 (25.5%), +2.3%
- Some RC4 suites enabled: 75,986 (51.2%), -1.3%
- Used with modern browsers: 34,660 (23.3%), -1.0%
Stream Ciphers

Key \rightarrow \text{key stream generator} \rightarrow Z_0, Z_1, Z_2, ... \rightarrow C_0, C_1, C_2, ...

Encryption: C_i = P_i \oplus Z_i,
Decryption: P_i = C_i \oplus Z_i,
i = 0, 1, 2, ...

Keystream randomness = plaintext security
• Rivest Code 4

• The most popular Stream Cipher for almost 30 years

• Details kept secret until the WEP attack in 2001
RC4 Algorithm

Key Scheduling Algorithm (KSA)

KSA(K):

- \( j = 0 \)
- \( S = [0, 1, 2, ..., 255] \)
- for \( i = 0..255 \)
  - \( j = (j + S[i] + K[i \text{ mode } L]) \)
  - \( S[i] \leftrightarrow S[j] \)

Pseudo-Random Generation Algorithm (PRGA)

PRGA(\( S_0 \)):

- \( i = 0 \)
- \( j = 0 \)
- \( S = S_0 \)
- While bytes are needed:
  - \( i = i + 1 \)
  - \( j = j + S[i] \)
  - \( S[i] \leftrightarrow S[j] \)
  - Emit \( S[S[i]+S[j]] \)

All operations are mod 256
RC4 Algorithm

KSA(K):
\[ j = 0 \]
\[ S = [0, 1, 2, \ldots, 255] \]
\[ \text{for } i = 0..255 \]
\[ j = (j + S[i] + K[i \text{ mode } L]) \]
\[ S[i] \leftrightarrow S[j] \]

PRGA(S\_0):
\[ i = 0 \]
\[ j = 0 \]
\[ S = S_0 \]
\[ \text{While bytes are needed:} \]
\[ i = i + 1 \]
\[ j = j + S[i] \]
\[ S[i] \leftrightarrow S[j] \]
\[ \text{Emit } S[S[i]+S[j]] \]
RC4 (In)Randomness

- RC4 in **NOT** pseudo-random
  - $2^{30}$ distinguisher (Fluhrer-McGrew, 2000) (patterns used in the RH attack)
  - $2^{26}$ byte distinguishing algorithm (Mantin, 2005)
  - $2^{45}$ Prediction algorithm (Mantin, 2005)
The weakest link of RC4 since 2001

Keystream biases
- The second-byte bias (Mantin-Shamir, 2001)
- Many others

Key-keystream correlations
- The IV Weakness and the WEP Attack (Fluhrer-Mantin-Shamir, 2001)
- Enhanced WEP Attack I (Mantin, 2005)
- Enhanced WEP Attack II (Tews-Weinmann-Pyshkin, 2007)
- More Key-keystream correlations (Klein, 2005)


The Invariance Weakness (Fluhrer-Mantin-Shamir, 2001)
The Invariance Weakness

- On TLS
- On RC4
- The Invariance Weakness
- The Attacks
- Conclusion
The Invariance Weakness

- The neglected counterpart of the IV Weakness
- Left in the shadows for 13 years

- RC4 weak keys
  - Huge class of keys ($2^{-24}$ fraction for 128bit keys)
  - Bad mixing of the key with the permutation
  - Permutation parts remain **intact**
Key Patterns

Pattern in Key

Pattern in Pseudo-random stream

Key Byte 0
Key Byte 1
Key Byte 2
Key Byte 3
Key Byte 12
Key Byte 13
Key Byte 14
Key Byte 15
Output Byte 0
Output Byte 1
Output Byte 2
Output Byte 3

MSBs
LSBs
MSBs
LSBs
The Weak Keys

• The keys (q-class)
  — K[i] = (1 - i) mod q
  — K[0] = 1

• How does it work?
  — Swaps preserve least significant bits
  — Initial permutation has S[i] = i (mod 2^q)
  — Final permutation has S[i] = i (mod 2^q)

KSA(K):
  j = 0
  S = [0, 1, 2, ..., 255]
  for i = 0..255
    j = (j + S[i] + K[i mode L])
    S[i] ↔ S[j]
Plaintext Leakage

- Initial permutation has LSB pattern
- LSB patterns leak to the keystream
  — But bad swaps ruin them
- Plaintext LSB leak

PRGA($S_0$):

```
i = 0
j = 0
S = S_0
```

While bytes are needed:

```
i = i + 1
j = j + S[i]
S[i] ↔ S[j]
Emit S[S[i]+S[j]]
```

Keystream randomness = plaintext security
# Weak Key Classes

<table>
<thead>
<tr>
<th># LSBs</th>
<th>Applicability</th>
<th>Class Fraction (8-byte key)</th>
<th>Class Fraction (16-byte key)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Keys with even number of bytes</td>
<td>$2^{-16}$</td>
<td>$2^{-24}$</td>
</tr>
<tr>
<td>2</td>
<td>Keys with number of bytes that is a multiple of 4</td>
<td>$2^{-23}$</td>
<td>$2^{-39}$</td>
</tr>
<tr>
<td>3</td>
<td>Keys with number of bytes that is a multiple of 8</td>
<td>$2^{-30}$</td>
<td>$2^{-54}$</td>
</tr>
<tr>
<td>4</td>
<td>Keys with number of bytes that is a multiple of 16</td>
<td>$2^{-37}$</td>
<td>$2^{-69}$</td>
</tr>
</tbody>
</table>
Plaintext Leakage

• When a weak key is used, “many” plaintext bit leak

• Q1: Can we tell when that happens?
  — Yes, when plaintext patterns exist

• Q2: How many bits?
Leakage Statistics
Diff-Based Leakage

- The permutation is ruined with the keystream generation
- Bit prediction gets out of sync when \( j \) hits a “ruined” part
- Switch to diff
Diff-Based Leakage (q=1)
Diff-Based Leakage (q=2)
Diff-Based Leakage (q=3)
The Leakage

- **Using the 1-Class**
  - 1\(^{st}\) diff LSB is guessed correctly with probability 0.68
  - 37\(^{th}\) diff LSB is guessed correctly with probability of 0.546
  - 100\(^{th}\) diff LSB is guessed correctly with probability of 0.503

- **Pattern tracking is possible for**
  - 37 bytes with 1/22 probability
  - 68 bytes with 1/64 probability
  - 100 bytes with 1/330 probability

- **First 100 LSBs are exposed to leakage**
The Attacks

- On TLS
- On RC4
- The Invariance Weakness
- The Attacks
- Conclusion
The Attacks

The Attack Scenario

- The Attack Scenario
- Using LSBs
- Man-in-the-Middle Attack
- Sniffing-Only Attack
- One-Time Encryption
RC4 @ TLS

Handshake Protocol

- Finished [36]
- Finished [36]

Record Protocol

- HTTP Request [n]

Upstream Key
Downstream Key

Use Bytes 0..35 of the Upstream keystream

Use Bytes 36..36+n of the Upstream keystream

Use Bytes 0..35 of the Downstream keystream

Use Bytes 36..36+n of the Downstream keystream
The Attack Basic Scenario

- Attacker waits for a “hit” - weak key occurrence
  - Attacker identifies the hit using plaintext patterns
  - $2^{24}$ keys until hitting a weak key
  - Several dozen/hundred hits to get successful tracking (target length dependent)

- Attacker predicts keystream LSB diffs
- Attacker recovers plaintext LSB values (after byte 36)
The Attacks

The Attack Scenario

- The Attack Scenario
- Using LSBs
- Man-in-the-Middle Attack
- Sniffing-Only Attack
- One-Time Encryption
LSB Leakage

• Acceleration of Trial and Error attacks
  — Sneak below threshold-based detectors

• Dictionary attack on weak passwords
## LSB for Weak Passwords

<table>
<thead>
<tr>
<th></th>
<th>Web Accounts</th>
<th>LSB Groups</th>
<th>Brute Force Worst Case</th>
<th>Brute Force Avg Case</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Top 100</strong></td>
<td>4.4%</td>
<td>68</td>
<td>6</td>
<td>1.5</td>
</tr>
<tr>
<td><strong>Top 1000</strong></td>
<td>13.2%</td>
<td>252</td>
<td>24</td>
<td>4</td>
</tr>
<tr>
<td><strong>Top 10,000</strong></td>
<td>30%</td>
<td>557</td>
<td>201</td>
<td>18</td>
</tr>
</tbody>
</table>
LSB for Credit Card Numbers

- CCN entropy:
  - 6-prefix: known
  - 4-suffix: not guarded
  - 1-byte: checksum

- With 16 LSBs, the search domain drops from 100,000 possibilities to only 1500
LSB for Session Cookies

- PHP Session Cookie: up-to $2^{32}$ brute-force reduction
- ASP Session Cookie: $2^{16}$ brute-force reduction
The Attacks

The Attack Scenario

- The Attack Scenario
- Using LSBs
- **Man-in-the-Middle Attack**
- Sniffing-Only Attack
- One-Time Encryption
Differences from BEAST/RH

- Attack requires a single “hit”
- 100 first bytes are at risk
- Extract only partial info
BEAST-like Attack

• 1 billion connections required
• Insensitive to Resets
Group Attack

Attack requires a single “hit”

Pool of Potential Victims
The Attacks

The Attack Scenario

- The Attack Scenario
- Using LSBs
- Man-in-the-Middle Attack
- Sniffing-Only Attack
- One-Time Encryption
Non-Targeted Passive Attack

Attack requires a single “hit”

Pool of Potential Victims
1 Billion Connections?

- Facebook has 890 million DAU (Daily Active Users)
- Most login more than once a day
The Attacks
The Attack Scenario

- The Attack Scenario
- Using LSBs
- Man-in-the-Middle Attack
- Sniffing-Only Attack
- One-Time Encryption
One-time Usage

• Every time you send a secret over TLS/RC4 connection
  — You have a 1:16 million chance to get a bad key
  — You have a 1 in a billion chance to get unlucky and leak a significant portion of your secret

• Small numbers, but definitely not negligible

• RC4 stats: 30% of Internet TLS connections
Conclusion

- On TLS
- On RC4
- The Invariance Weakness
- The Attacks
- Conclusion
Summary

- The Invariance Weakness of RC4 can be used to mount new attacks on TLS

- The Reset Insensitivity nature of the attack opens the door to new attack scenarios

- First passive attack on TLS
Conclusions

- RC4 is not a secure cipher (old news)
- The initialization mechanism of RC4 is very weak (old news)
- The impact of these facts on the (In)Security of systems using RC4 is underestimated