TLS attacks and stream ciphers

Sec-heads
Linköping
12-13 Februari 2013

Simon Josefsson
simon@josefsson.org

Overview

- Motivation - TLS attacks
  - Padding oracles
  - BEAST
  - CRIME
  - Lucky-13
- RC4
- Salsa20

Padding oracles

- Side channel against CBC-PAD
- Attack described in 2002 by Serge Vaudenay
- Demo in 2010 by Rizzo and Duong

Padding oracles

- Attack uses information about whether plaintext padding was the expected
- Guess last byte of ciphertext → if pad correct, then you can infer plaintext using xor, if pad incorrect → guess again
- WTLS broke but TLS didn't
  - TLS encrypts errors, and disconnect sessions
- Solutions: Don't return pad error, always MAC, use Pad-Encrypt-then-MAC, use non-CBC

TLS BEAST Attack

- ”Browser Exploit Against SSL/TLS”
- Attack described in 2002 by Phil Rogaway, extended to IPSEC/SSH, led to TLS 1.1/1.2
- Demo by Duong and Rizzo in 2011

TLS BEAST Attack

- Chosen-plaintext with known/predicted IV
- The CBC IV for each record (except first) is the previous records' last ciphertext
- Arrange for known plaintext to be combined with one character of unknown data in one block
- Typical attack: Decrypt cookies
- Solution: TLS 1.1/1.2, send empty MAC'ed record (some interop issues), use non-CBC

TLS CRIME Attack

- ”Compression Ratio Info-leak Made Easy”
- Plaintext injection combined with information leakage via side channel (size)
- Attack described in 2002 by John Kelsey
- Obvious from a theoretical point of view
  → not obvious how to use it to attack
- Demo by Duong and Rizzo in 2011

TLS CRIME Attack

- Observe size of packet while crafting new requests → payload contains both secret (cookie) and known plaintext
- Idea: when encrypted traffic is shorten, the known plaintext is similar to the crafted data. Divide/Conquer
- Solution: disable compression

TLS Lucky-13 Attack

- Multi-session attack – plaintext in same position, client re-try many times
- Extension of the oracle padding attack: instead of error message, use timing to infer error condition

TLS Lucky-13 Attack

- RFC 5246:

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. For instance, if the pad appears to be
incorrect, the implementation might assume a zero-length pad and then
compute the MAC. This leaves a small timing channel, since MAC
performance depends to some extent on the size of the data fragment,
but it is not believed to be large enough to be exploitable, due to
the large block size of existing MACs and the small size of the
timing signal.

TLS Lucky-13 Attack

- Solution: Use AEAD ciphers, use stream cipher, remove timing side-channel for CBC
- Non-solution: always MAC regardless of pad errors isn't enough – CBC-PAD is hard!

RC4

```
i := 0
j := 0
while GeneratingOutput:
    i := (i + 1) mod 256
    j := (j + S[i]) mod 256
    swap values of S[i] and S[j]
    K := S[(S[i] + S[j]) mod 256]
    output K
endwhile
```

RC4

- Really fast - ~700MByte/second
- Considered broken [by crypto people]
- ”Everyone” uses it – Google, etc
- SSLabs.com ”TLS Deployment Best Practices” - February 2012
  - ”Practical mitigation requires that your servers speak only RC4 when using TLS v1.0 or SSL v3.”
- Why is there no competition?

Salsa20

- Modern stream cipher - Daniel J. Bernstein
- Fast – 300 Mbyte/s on my laptop
- Built on a hash function (not necessarily a cryptographic one)
- Stream is concatenated hashes of key, 64-bit nonce and a 64-bit block number
- Stream is seekable → works with DTLS
- Trivial setup costs

Stream ciphers in TLS

- Only RC4 is defined
- DTLS disallow RC4 because it is state-full
- Problem: no place to put the nonce

Solutions

- Solution 1: Abuse existing GenericStreamCipher and pre-pend nonce
- Solution 2: Abuse GenericAEADCipher and add MAC
- Solution 3: Add another cipher type (TLS 1.3?)

Future?

- Authenticated stream ciphers
  - Interesting but difficult to standardize at this point