# Plaintext-Recovery Attacks Against Datagram TLS

*Nadhem Alfardan and Kenneth Paterson*

**Information Security Group**
Royal Holloway, University of London

6th Feb 2012

# Contents

# Results

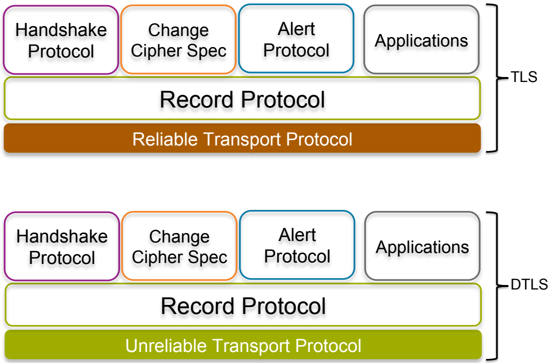
Plaintext-recovery attacks through which we were able to:

- Decrypt arbitrary amount of ciphertext in the case of the OpenSSL implementation of DTLS.
- Decrypt the four most significant bits of the last byte in every block in the case of the GnuTLS implementation of DTLS.

# Introduction to DTLS

- Datagram Transport Layer Security (DTLS) was first introduced in NDSS 2004.
- IETF assigned RFC 4347 to DTLS 1.0 in 2006. RFC 6347 updates RFC 4347 and was published in Jan 2012 under DTLS 1.2.
- By design, DTLS 1.0 is very similar to TLS 1.1. RFC 4347 presents only the changes to TLS 1.1 and refers to RFC 4346 for the rest of the specification.
- A number of RFC documents have been published on DTLS.
- DTLS is used in a number of implementations.

- DTLS runs over an unreliable protocol such as Unreliable Datagram Protocol (UDP).



Changes to TLS 1.1 also include:

- Implementations of DTLS should **silently discard** data with bad MACs or padding. No error messages are generated in both cases.
- In DTLS, connections are **not** terminated in the case of an error.
- In DTLS, fragmentation of record messages is not permitted.
- DTLS optionally supports record replay protection.

*There are other changes, but they are not of relevance.*

# Previous Attacks

- Vaudenay's padding oracle, ($\mathcal{PO}$) applies to CBC-mode encryption.
- $\mathcal{PO}$ returns `VALID` if the padding is correct and `INVALID` otherwise.
- The realisation of this oracle relies on the attacker having access to TLS error messages; `decryption_failed` and `bad_record_mac` which are classified as fatal.
- In the case of TLS 1.0, both of these error messages are encrypted.
- Connections are terminated immediately whenever such errors are encountered.

**Algorithm 1:** Decrypting a block using a padding oracle $\mathcal{PO}$ for TLS/DTLS.

**Data**: $C^*_{t-1}$, $C^*_t$
**Result**: $P^*_t = D_k(C^*_t) \oplus C^*_{t-1}$
Let $R$ be a random $b$-byte block.;
**for** $i = 0$ **to** $b - 1$ **do**
  **for** $byte = 0$ *to* 255 **do**
    $R[i] = \texttt{byte}$;
    $C = R || C^*_t$;
    **if** $\mathcal{PO}(C) = \texttt{VALID}$ **then**
      $P[i] = R[i] \oplus C^*_{t-1}[i] \oplus i$;
      Break;
  **for** $j = 0$ *to* $i$ **do**
    $R[j] = R[j] \oplus (i) \oplus (i+1)$;
Output $P$;

- The work of Canvel *et al.* exploits the fact that processing a message with valid padding may take longer than the processing of a message with invalid padding:
  - The timing difference comes from the MAC verification process.
- Canvel *et al.* were able to extract fixed plaintext in the form of TLS-encrypted passwords. Connections had to be re-established after being terminated, making the attack difficult to implement.
- Countermeasures were introduced in TLS 1.1:
  - One of them is to perform MAC verification on packets that fail the padding check.

# Padding Oracle Realisation Against OpenSSL

- DTLS Packets with invalid padding are silently discarded and MAC verification is not performed. **No** error messages are generated when the padding error is encountered.
  - This protects the system from the attack introduced by Canvel *et al.*
- We constructed a new realisation for the padding oracle to exploit the OpenSSL implementation of DTLS.

**Algorithm 2:** Padding Oracle for OpenSSL implementation of DTLS

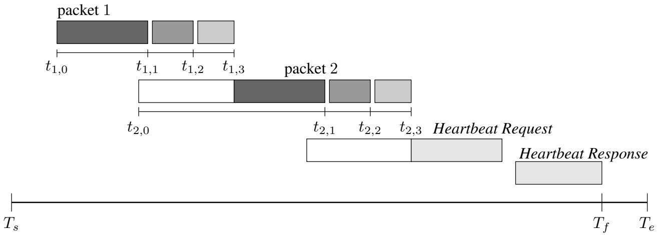
**Data**: $C$
**Result**: `VALID` or `INVALID`
**for** $q = 1$ **to** $m$ **do**
  $RTT_q$ = **Timer(**$C$**)**;

$RTT$=Mean($RTT_1, RTT_2, \ldots, RTT_m$);
**if** $RTT \geq T$ **then**
  **return** `VALID`;
**else**
  **return** `INVALID`;

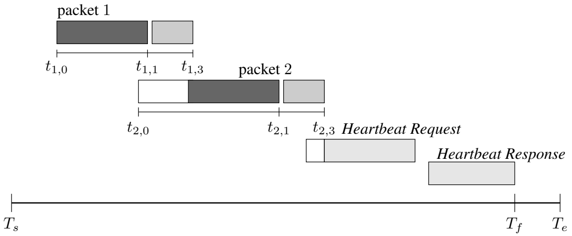
**Timer(**$C$**)**
Set $T_s$ = current time;
Send $n$ copies of $P_C$, a DTLS packet containing $C$, to the targeted system;
Send a Heartbeat request packet to the targeted system;
Set $T_e$ = time when Heartbeat response packet is seen;
**return** $(T_e - T_s)$

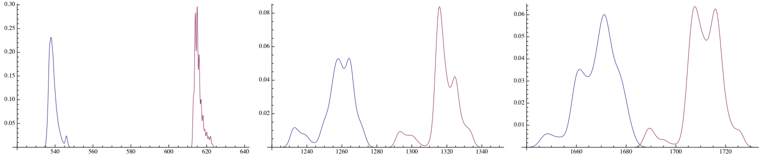
- We were able to use Heartbeat messages to compensate the lack of error messages. The advisory sends a Heartbeat request message right after the attack message(s).
- The advisory calculates the time from sending the first message to receiving the Heartbeat response message.
- To amplify the timing difference we used a train of packets.

Time-line for packet train with valid padding



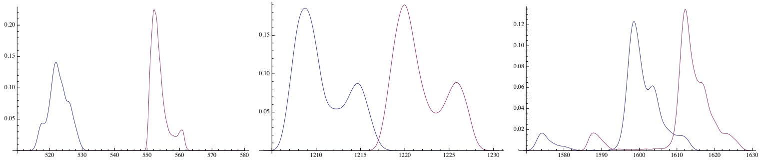

Time-line for packet train with invalid padding






(a) $l = 256$ (b) $l = 1024$ (c) $l = 1456$

Figure: 3DES – PDFs for trains of 10 packets and varying the DTLS payload length, $l$.




(a) $l = 256$ (b) $l = 1024$ (c) $l = 1456$

Figure: AES-256 – PDFs for trains of 10 packets and varying the DTLS payload length, $l$.

| *n* and *l* | **128** | **160** | **192** | **224** | **256** | **288** |
|---|---|---|---|---|---|---|
| **1** | 0.99 | 0.99 | 1.00 | 0.99 | 1.00 | 0.99 |
| **2** | 0.99 | 1.00 | 0.99 | 1.00 | 1.00 | 0.98 |
| **5** | 0.99 | 1.00 | 1.00 | 1.00 | 1.00 | 0.98 |
| **10** | 0.98 | 1.00 | 0.99 | 1.00 | 1.00 | 0.99 |
| **20** | 0.99 | 0.99 | 1.00 | 1.00 | 1.00 | 0.99 |
| **50** | 0.99 | 0.99 | 1.00 | 1.00 | 0.98 | 0.95 |

Table: Success probabilities per byte for AES, for various attack parameters (with anti-replay disabled).

*n* is the train size and *l* is the DTLS payload size in bytes.

# Attacking the GnuTLS Implementation of DTLS

- Unlike OpenSSL, GnuTLS share the same code for TLS and DTLS.
- GnuTLS implements the fix introduced in TLS 1.1 and hence is not vulnerable to our attack against OpenSSL.
- We were able to recover the four most significant bits of the last byte in each ciphertext block by exploiting a different issue in the code and using the same technique.

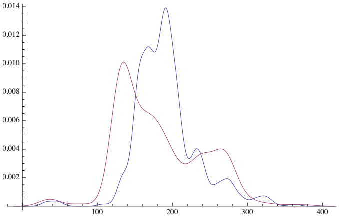



Figure: PDFs for AES-256 with HMAC-SHA256, $l = 176$, $n = 5$, based on 1000 trials, with outliers removed.

# Fixes

- On 4th of Jan 2012, OpenSSL issued releases 1.0.0f and 0.9.8s which included a fix.
- On 6th of Jan 2012, GnuTLS issued release 3.0.11 which included a fix.

# Lessons

- Lack of error messages does not necessarily mean that the system is not vulnerable.
- Although the GnuTLS implementation of DTLS follows the standard, we were able to deploy similar techniques to attack the implementation and recover a limited amount plaintext.
- Features of lower layer protocols can have a major influence on security at higher layers.