TLS Under Siege: A Bug Hunter's Perspective

Neel Mehta, Google Security

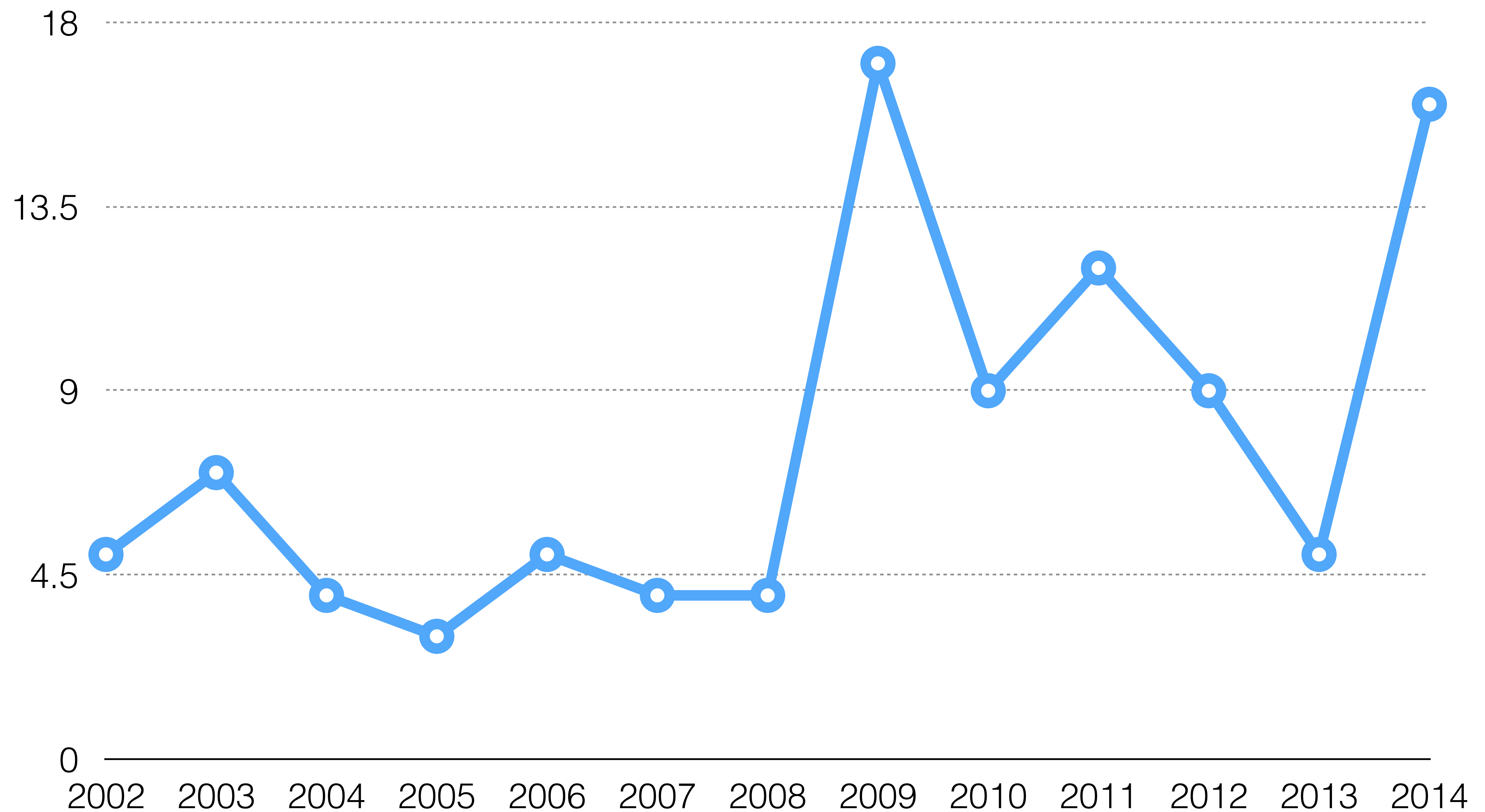
What I'll Cover

- Lightning-fast background on TLS.
- A bunch of TLS bugs, both new and old.
- Perspective gained from reading lots of code.
    - How bad are the implementations?
    - Are some worse than others?
    - Why? Where?

Overview

- Encryption matters more, and must be done at scale.
  - Drives protocol changes, and implementation changes.
- What's the health of our implementations?
  - Where are the weak spots? Why?
  - Lots of looking at code snippets (audience participation highly encouraged).

OpenSSL CVE's Assigned Per Year - 2002 - 10/2014



Memory Corruption

- Much less common today.
- Used to be more common:
- From OpenSSL advisory in July 2002:

All four of these are potentially remotely exploitable.

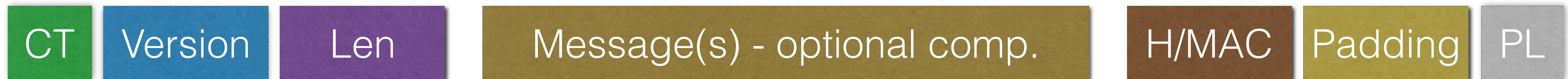
```
1. The client master key in SSL2 could be oversized and overrun a
   buffer. This vulnerability was also independently discovered by
   consultants at Neohapsis (http://www.neohapsis.com/) who have also
   demonstrated that the vulerability is exploitable. Exploit code is
   NOT available at this time.

2. The session ID supplied to a client in SSL3 could be oversized and
   overrun a buffer.

3. The master key supplied to an SSL3 server could be oversized and
   overrun a stack-based buffer. This issues only affects OpenSSL
   0.9.7 before 0.9.7-beta3 with Kerberos enabled.

4. Various buffers for ASCII representations of integers were too
   small on 64 bit platforms.
```

TLS Stacks

- I reviewed the following:
  - OpenSSL, and forks (BoringSSL, libressl)
  - NSS
  - GnuTLS
  - PolarSSL
  - Secure Transport (iOS / OS X)
  - Schannel 7 (MS)
  - Botan

TLS Protocols

TLS Record Layer

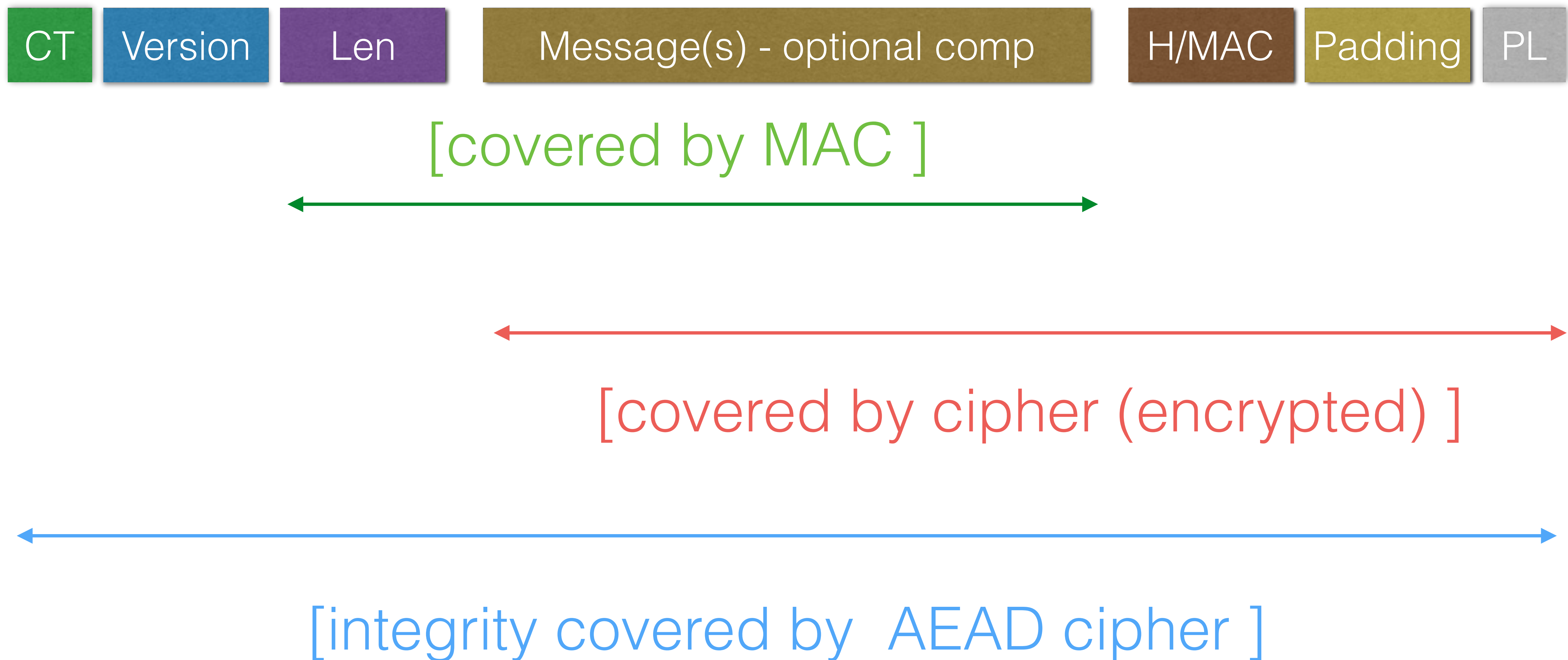


Record Layer Content / Protocol Types:

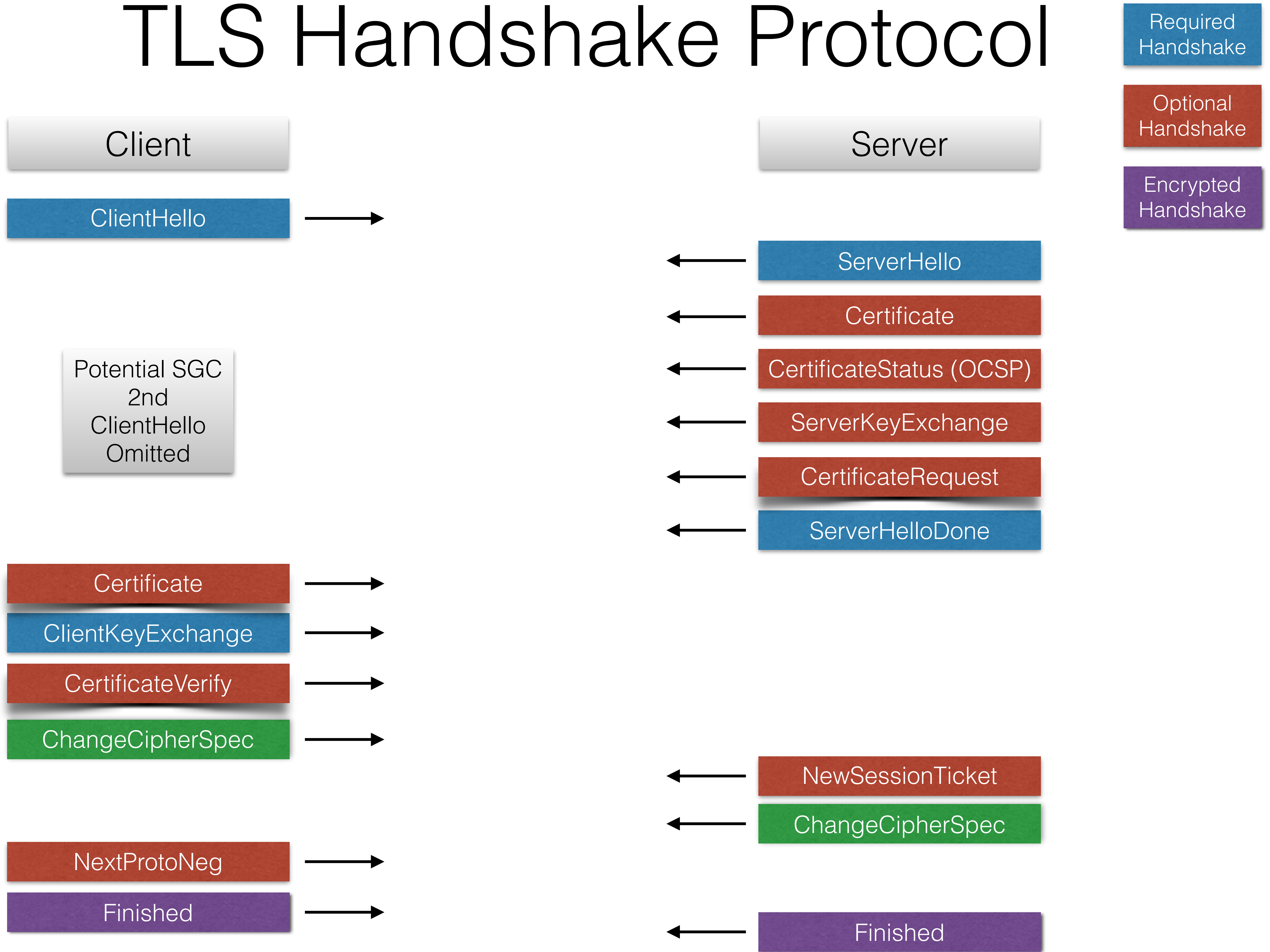
- Handshake - agree on cipher state
- ChangeCipherSpec - switch cipher states
- Alert - report fatal and warning errors
- Heartbeat - ?!?

TLS Protocols

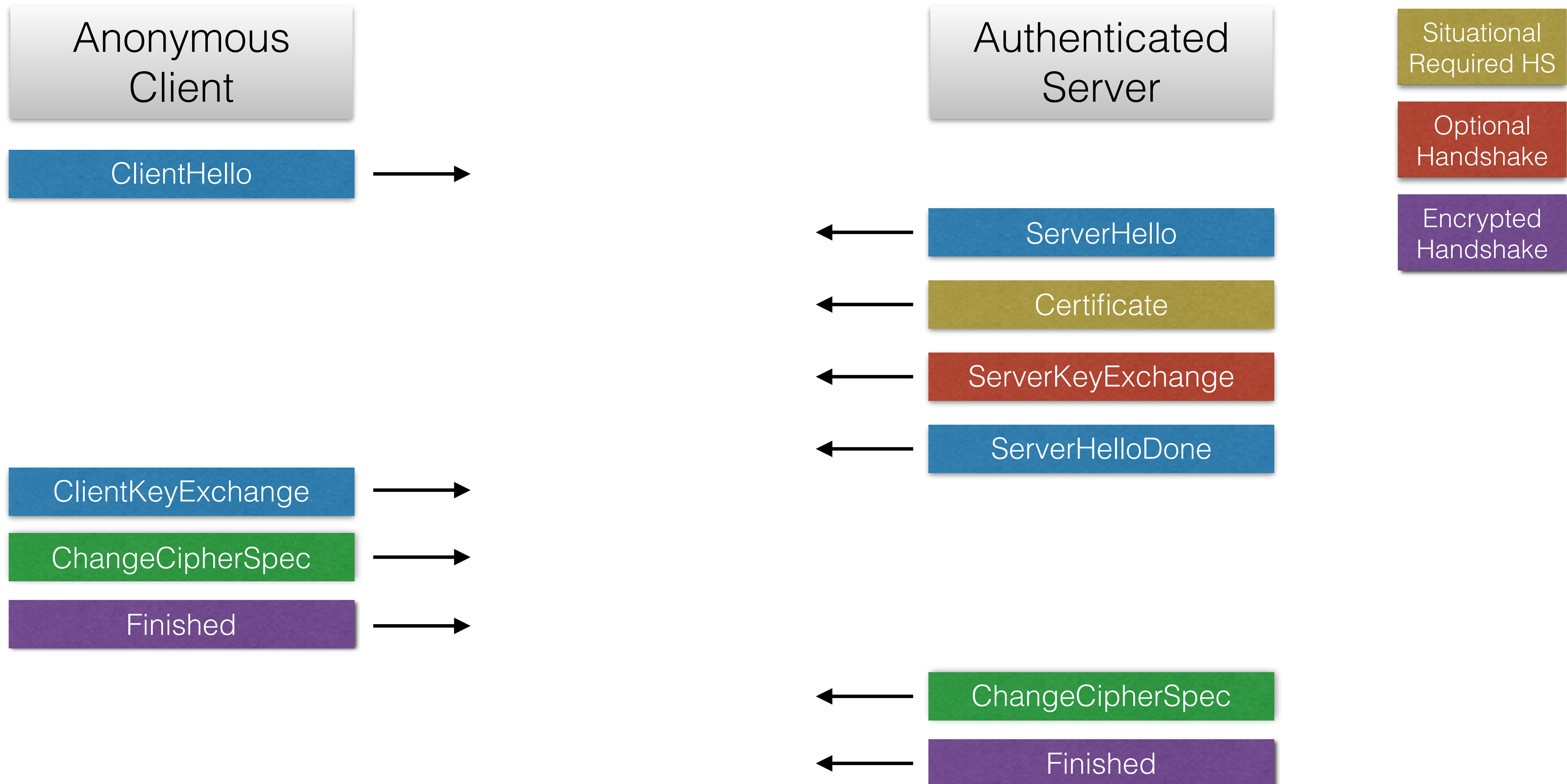
TLS Record Layer



TLS Handshake Protocol



Typical Handshake



Handshake Protocol Goals

- Agree upon:
  - Protocol versions.
  - Ciphers / MAC algorithms.
  - Compression.
- Authenticate server or client.
- Symmetric key state.
- Assurances against tampering.

PKI / X.509

- Bind identities to public keys.
  - Handle revocation.
  - X.509 path construction.
  - Roots of trust (trusted root CA store).
- Asymmetric encryption to establish an authenticated pre-master-secret.

Master Secret - PRF’s

- MasterSecret = PRF (Pre-Master Secret + “master secret” + ClientRandom + ClientOpaquePRFInputs + ServerRandom + ServerOpaquePRFInputs) [0..47]

- KeyBlock = PRF (MasterSecret + “key expansion” + ClientRandom + ServerRandom)[0..N]

- Gives: IV’s, Server / Client MAC secrets, Server / Client write keys.

Handshake Protocol Protection

- CCS record protocol message - switches to pending, agreed-upon symmetric key block.
- FinishedMessage:
  - Calculate finished hash
    - PRF (MasterSecret + (“client finished” || “server finished”) + **MD5(handshake messages) + SHA1(handshake messages)**)[0..N]
    - For TLS 1.2 - SHA-2 (usually SHA-256) instead (sigalgs).
- Send encrypted with cipher state, verify the finished message from the other side.
  - Assurances against tampering at HS layer.

Types of Implementation Errors

- RNG seeding bugs:
    - Debian / static analysis.
- Memory Corruption.
- Integer Safety.
- Information Leakage - explicit and implicit.
- Timing Issues.
- State Issues.
- Dangerous code constructs.

Protocol Bugs

- Interesting protocol flaws: Bleichenbacher attack, BEAST, CRIME, Lucky13.
    - Padding oracles, timing attacks, predictable IV's, cipher biases.
  - Switch from CBC -> RC4 -> CBC.
- Strongly driving implementation and protocol changes, occasionally errors..

Relative Structure

- Systematic bounds checks on read, writes?
- Threading.
- API Design Choices.
- Memory management.
- Parallel assembly implementations.

Protocol Influences on Implementation

- What properties of SSL tend to influence implementations?

Attack Surface

- Properties of the TLS protocol cause issues:
    - Layered Protocols.
    - Compression - usually disabled (now).
    - Encryption / MAC / HMAC.

Attack Surface - Necessities

- Handshake record processing - a huge state machine:
  - Deserialize, re-serialize ping-pong.
  - Re-use the same buffers, ad-infinitum.
- Authentication (client and server).
- Concurrency.
- PKI / X.509.
- ASN.1

Attack Surface - ASN.1

- ASN.1 more exposed than it used to be:
  - Also more fuzzed than ever.
- Server-side:
  - OCSP extensions / stapled responses, client-certificates, Kerberos.
- Client-side:
  - X.509 certificates, OCSP, CRL’s.

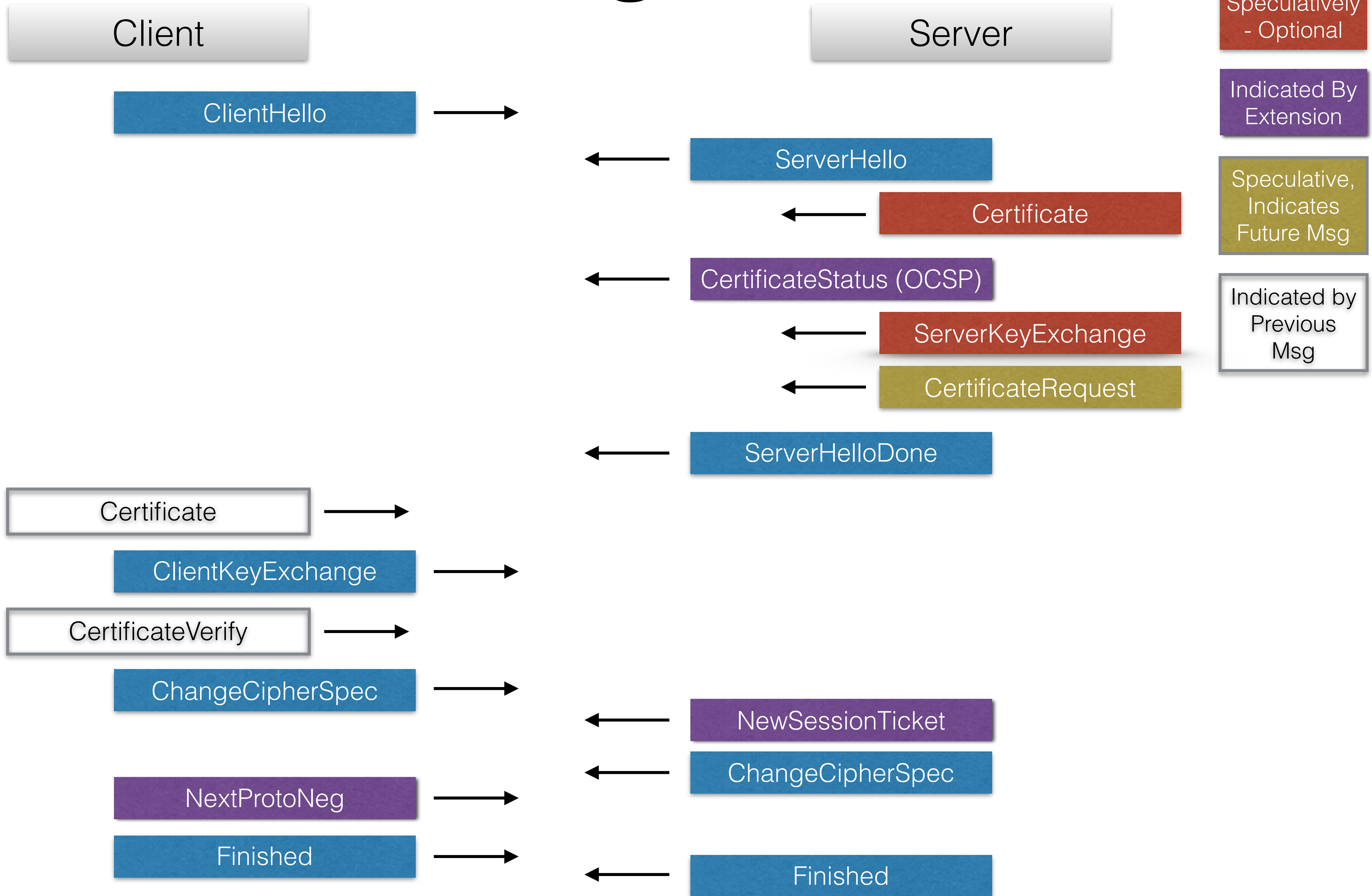
Layered Protocols

- Record protocol encapsulates other protocols:
  - Fragmentation.
  - Buffering.
  - Protocol priorities.

Buffering Bugs

- Many handshake protocol messages are optional.
- Handshake protocol message order depends on:
  - Protocol versions.
  - TLS extensions.
  - Client / Server authentication.
  - Negotiated cipher suite and cert pub-key type.

HS Msg Order



OpenSSL HS Message Buffering API

```
long (*ssl_get_message)(
    SSL *s, int st1, int stn, int mt, long max, int *ok);
```

- ‘max’ indicates the largest HS message to read.
- If ‘mt’ >= 0, then enforce that the message read is of type ‘mt’.
- 2 ‘output’ parameters to indicate success:
  - Returns ‘length’ of message, or negative length on error.
  - Sets ‘*ok’ to non-zero to indicate success
- Issues:
  - Caller must re-call for bad speculative reads (unexpected HS message types).
  - Discerning success / failure is multiple steps.

OpenSSL HS Message Buffering Code

```
long ssl3_get_message(SSL *s, int st1, int stn, int mt, long max, int *ok)
  {
  unsigned char *p;
  unsigned long l;
  long n;
  int i,al;

  if (s->s3->tmp.reuse_message)
    {
    s->s3->tmp.reuse_message=0;
    if ((mt >= 0) && (s->s3->tmp.message_type != mt))
      {
      al=SSL_AD_UNEXPECTED_MESSAGE;
      SSLerr(SSL_F_SSL3_GET_MESSAGE,SSL_R_UNEXPECTED_MESSAGE);
      goto f_err;
      }
    *ok=1;
    s->init_msg = s->init_buf->data + 4;
    s->init_num = (int)s->s3->tmp.message_size;
    return s->init_num;
    }
...
```

API Implications - OpenSSL HS Message Buffering

- Fails to properly enforce ‘max’ when re-reading a buffered message:
  - Checked against the ‘max’ from the first call, which may be different.
- Oops, however:
  - OpenSSL doesn’t explicitly rely on enforcement of ‘max’.
    - At least not today (maybe DTLS?).

API Implications - Multiple Success / Failure Conditions

```
long (*ssl_get_message)(
    SSL *s, int st1, int stn, int mt, long max, int *ok);
```

- To check for success, caller must validate several things:
  - '*ok' != 0
  - Return value 'length' >= 0.
  - Expected HS message fits within bounds of 'length' returned.

OpenSSL OOB Reads

- ssl3_get_client_hello() - Note that zero-length handshake messages are legal:

```
n=s->method->ssl_get_message(s,
SSL3_ST_SR_CLNT_HELLO_B,
SSL3_ST_SR_CLNT_HELLO_C,
SSL3_MT_CLIENT_HELLO,
SSL3_RT_MAX_PLAIN_LENGTH,
&ok);

if (!ok) return((int)n);
s->first_packet=0;
d=p=(unsigned char *)s->init_msg;
/* use version from inside client hello, not from record header
 * (may differ: see RFC 2246, Appendix E, second paragraph) */
s->client_version=(((int)p[0])<<8)|(int)p[1];
p+=2;
```

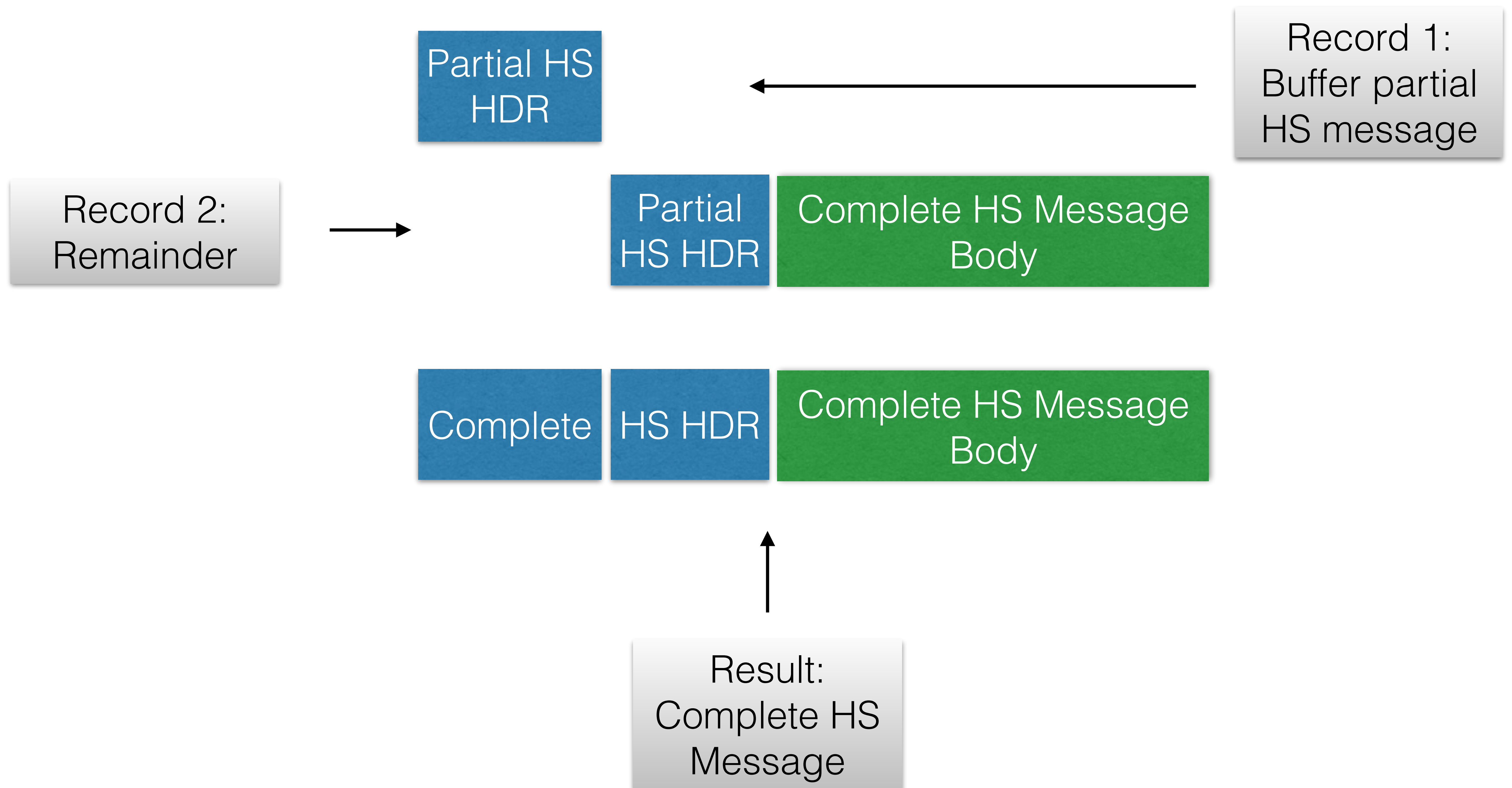
Buffering / Fragmentation Bugs - Secure Transport

```
OSStatus
SSLProcessHandshakeRecord(SSLRecord rec, SSLContext *ctx)
{   OSStatus        err;
    size_t          remaining;
    UInt8           *p;
  UInt8     *startingP;   // top of record we're parsing
    SSLHandshakeMsg message = {};
    SSLBuffer       messageData;

    if (ctx->fragmentedMessageCache.data != 0)
    {

    [ Concatenate fragment cached record + new record data ]

   }
    else
    {   remaining = rec.contents.length;
        p = rec.contents.data;
    }
  startingP = p;
```

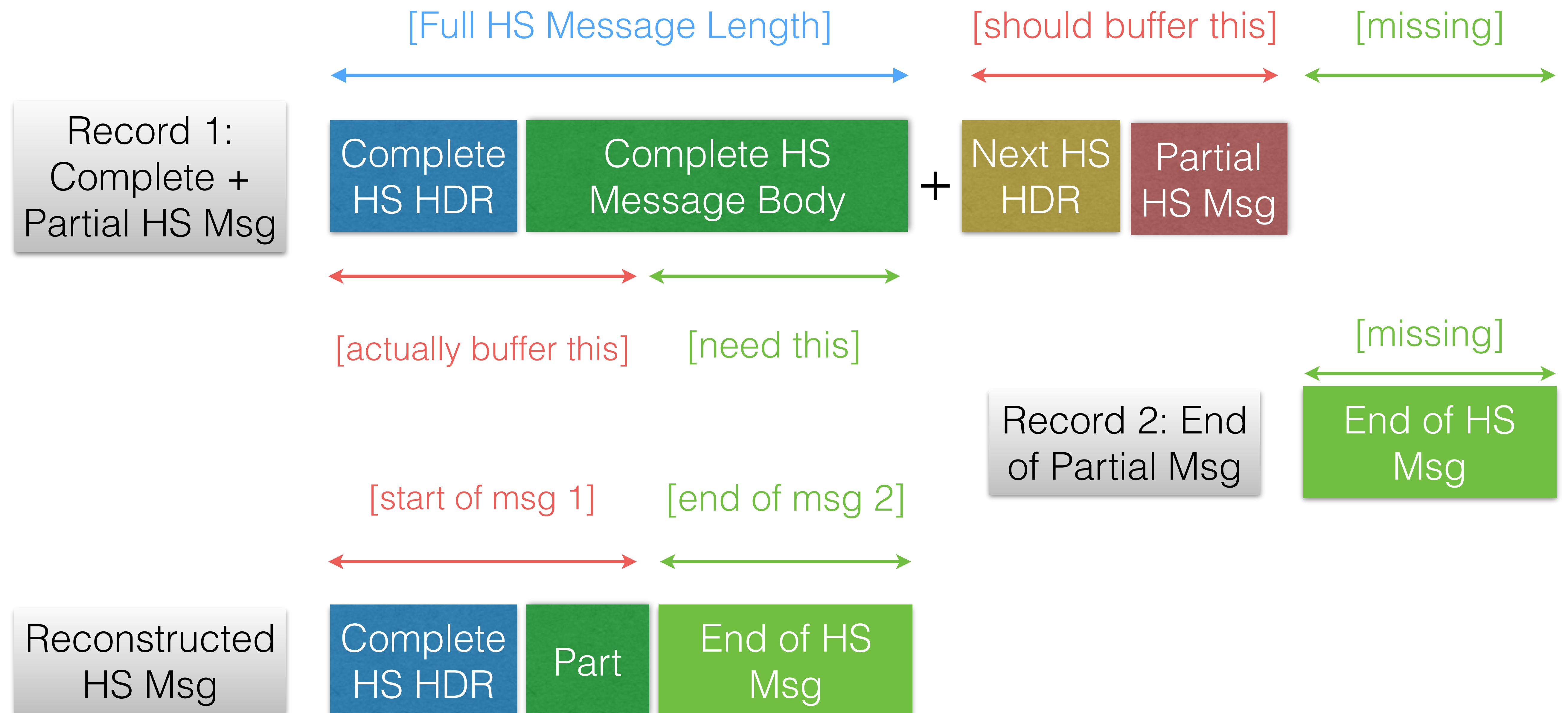
Buffering / Fragmentation Bugs - Secure Transport

```
size_t head = 4;

while (remaining > 0)
{
    if (remaining < head)
        break;  /* we must have at least a header */

    messageData.data = p;
    message.type = (SSLHandshakeType)*p++;
    message.contents.length = SSLDecodeSize(p, 3);


    p += 3;

    if ((message.contents.length + head) > remaining)
        break;

    [ Have at least one complete handshake message, advance 'p', subtract from 'remaining',
      process message, advance handshake ]
}

if (remaining > 0)      /* Fragmented handshake message */
{   /* If there isn't a cache, allocate one */
    if (ctx->fragmentedMessageCache.data == 0)
    {   if ((err = SSLAllocBuffer(&ctx->fragmentedMessageCache, remaining)))
        {   SSLFatalSessionAlert(SSL_AlertInternalError, ctx);
            return err;
        }
    }
        if (startingP != ctx->fragmentedMessageCache.data)
        {   memcpy(ctx->fragmentedMessageCache.data, startingP, remaining);
            ctx->fragmentedMessageCache.length = remaining;
        }
}
```

Intent: Handle a Single, Partial Handshake Message Record



Alternate Result: Record Containing Multiple HS Msgs



Implications

- Definitely not as-intended.
  - Implications aren’t immediately clear to me
    - Crypto-help needed.
  - ‘Just fix it’.

Different width sizes

- Record protocol: UINT16
- TLS extensions: UINT16
- Handshake protocol: UINT24
- More code snippets!

Different length widths - Potential Bug in Secure Transport

- TLS extension total length field: UINT16 - max 0xFFFF
- Handshake message length: UINT24
- Potential bug in SSLEncodeClientHello():
    - Add up various TLS extension sizes, setting individual length field.
    - Check if sum of TLS ext lengths > 0xffff
        - If too large, zero out all extension lengths
            - Indicates to not add any extensions.
            - Also don’t allocate any space for them in ClientHello.

Different length widths - Potential Bug in Secure Transport

```
OSStatus
SSLEncodeClientHello(SSLRecord *clientHello, SSLContext *ctx)
{

  size_t      sessionTicketLen = 0;
  size_t      serverNameLen = 0;
  size_t      pointFormatLen = 0;
  size_t      suppCurveLen = 0;
  size_t      signatureAlgorithmsLen = 0;
  size_t      totalExtenLen = 0;
…
    /* RFC 5746: We add the extension only for renegotiation ClientHello */
    if(ctx->secure_renegotiation) {
        totalExtenLen += 2 + /* extension type */
                         2 + /* extension length */
                         1 + /* lenght of renegotiated_conection (client verify data) */
                         ctx->ownVerifyData.length;
    }
…
  if(ctx->sessionTicket.length) {
    sessionTicketLen = 2 +  /* extension type */
               2 + /* 2-byte vector length, extension_data */
               ctx->sessionTicket.length;
    totalExtenLen += sessionTicketLen;
  }
…
```

Different length widths - Potential Bug in Secure Transport - cont'd

```
if(totalExtenLen != 0) {
  /*
   * Total length extensions have to fit in a 16 bit field...
   */
  if(totalExtenLen > 0xffff) {
    sslErrorLog("Total extensions length EXCEEDED\n");
    totalExtenLen = 0;
    sessionTicketLen = 0;
    serverNameLen = 0;
    pointFormatLen = 0;
    suppCurveLen = 0;
          signatureAlgorithmsLen = 0;
  }
  else {
    /* add length of total length plus lengths of extensions */
    length += (totalExtenLen + 2);
  }
}

  clientHello->contentType = SSL_RecordTypeHandshake;
  head = SSLHandshakeHeaderSize(clientHello);
  if ((err = SSLAllocBuffer(&clientHello->contents, length + head)))
      goto err_exit;
…
```

Different length widths - Potential Bug in Secure Transport - cont'd

```
/*
 * Append ClientHello extensions.
 */
if(totalExtenLen != 0) {
  /* first, total length of all extensions */
  p = SSLEncodeSize(p, totalExtenLen, 2);
}

  if(ctx->secure_renegotiation){
      assert(ctx->ownVerifyData.length<=255);
      p = SSLEncodeInt(p, SSL_HE_SecureRenegotation, 2);
      p = SSLEncodeSize(p, ctx->ownVerifyData.length+1, 2);
      p = SSLEncodeSize(p, ctx->ownVerifyData.length, 1);
      memcpy(p, ctx->ownVerifyData.data, ctx->ownVerifyData.length);
      p += ctx->ownVerifyData.length;
  }

if(sessionTicketLen) {
  sslEapDebug("Adding %lu bytes of sessionTicket to ClientHello",
    ctx->sessionTicket.length);
    p = SSLEncodeInt(p, SSL_HE_SessionTicket, 2);
  p = SSLEncodeSize(p, ctx->sessionTicket.length, 2);
  memcpy(p, ctx->sessionTicket.data, ctx->sessionTicket.length);
  p += ctx->sessionTicket.length;
}
```

Size Semantics

- Lengths are # of bytes, not number of elements.
- Many fields have a minimum length, or length must be even multiples of element size.
  - Cipher suites (2).
  - Server certificate lengths (3).
  - Signature algorithm (2).
  - Use srtp extension - DTLS (2)

OpenSSL Size Semantics

- From ssl3_get_server_certificate():

```
 n=s->method->ssl_get_message(s,
   SSL3_ST_CR_CERT_A,
   SSL3_ST_CR_CERT_B,
   -1,
   s->max_cert_list,
   &ok);

  if (!ok) return((int)n);
…
 p=d=(unsigned char *)s->init_msg;
…
  n2l3(p,llen);
  if (llen+3 != n)
    {
    al=SSL_AD_DECODE_ERROR;
    SSLerr(SSL_F_SSL3_GET_SERVER_CERTIFICATE,SSL_R_LENGTH_MISMATCH);
    goto f_err;
    }
  for (nc=0; nc<llen; )
    {
    n2l3(p,l);
    if ((l+nc+3) > llen)
      {
      al=SSL_AD_DECODE_ERROR;
      SSLerr(SSL_F_SSL3_GET_SERVER_CERTIFICATE,SSL_R_CERT_LENGTH_MISMATCH);
      goto f_err;
      }
```

Other OpenSSL OOB Reads

- Client:
  - ssl3_get_key_exchange()
  - ssl3_get_certificate_request()
- Server:
  - ssl3_get_client_hello()
  - ssl3_get_client_key_exchange()
- Server with client auth enabled:
  - ssl3_get_cert_verify()
  - ssl3_get_client_certificate()
- DTLS client:
  - dtls1_get_hello_verify()

Interoperability

- Workaround for buggy clients, mainly.
- Protocol version differences:
  - Wire protocol differences.
- Field duplication.
  - Especially protocol versions.
- State machine changes (early CCS bug + NextProtoNegotiation = key state without master secret).

Coding Styles

- Repeated constructs with inherent risk:
  - OpenSSL ‘goto and free’ constructs:

```
char* ptr = NULL;
…
ptr = OPENSSL_malloc(size);
…
if (error) {
  goto err;
}
…
err:
  if (ptr != NULL) {
    free(ptr);
    ptr = NULL;
  }
```

  - Easy to forget outstanding references.
  - Recursion vs. unbounded recursion.

State Issues - OpenSSL DTLS

- Just terrible…
- Three bugs in OpenSSL's DTLS fragment handling, within lines of each other in 2 related functions
  - DTLS recursion crash: CVE-2014-0221
  - DTLS memory corruption CVE-2014-0195 - memory corruption via large second message length
  - DTLS double free: CVE-2014-3505

- DTLS memory corruption CVE-2014-0195

- Caused by duplication of fields in fragmented messages, and protocol layering / buffering.
  - Each DTLS message has fields:
    - seq
      - Consecutive sequence numbers identify related messages.
    - msg_len
      - Should be the same across all fragments of a message.
    - frag_off
    - frag_len

Redundant Fields

- Attack:
  - First buffer a fragment with a short ‘msg_len’.
  - Then transmit a second fragment (seq + 1), with a longer ‘msg_len’ than first.

State Issues

- Sometimes exacerbated by wire-protocol compatibility between related protocols.
  - DTLS1 and TLS1.x are share many common handshake messages.
    - DTLS and TLS in OpenSSL, client and server, all vulnerable to Heartbleed.
  - Some fields / messages are only TLS or DTLS though:
    - Certain HS messages, TLS extensions.
  - TLS is one giant state machine - is there any protocol crossover?

SecureTransport - Redundant Fields + Protocol Crossover

- SSLProcessServerHelloVerifyRequest()
- Should be a DTLS-only message:

```
#if ENABLE_DTLS
        case SSL_HdskHelloVerifyRequest:
            if (ctx->protocolSide != kSSLClientSide)
                goto wrongMessage;
            if(ctx->state != SSL_HdskStateServerHello)
                goto wrongMessage;
            /* TODO: Do we need to check the client state here ? */
            err = SSLProcessServerHelloVerifyRequest(message.contents, ctx);
            break;
#endif
…
    p = message.data;

    protocolVersion = (SSLProtocolVersion)SSLDecodeInt(p, 2);
    p += 2;

    /* TODO: Not clear what else to do with protocol version here */
    if(protocolVersion != DTLS_Version_1_0) {
        sslErrorLog("SSLProcessServerHelloVerifyRequest: protocol version error\n");
        return errSSLProtocol;
    }
```

Modern TLS Stack Memory Corruption

- Tend to be found in obscure parts of the protocol.

- OpenSSL SRP overflow: CVE-2014-3512

```
  BIGNUM *SRP_Calc_u(BIGNUM *A, BIGNUM *B, BIGNUM *N) {
  …
  int longN= BN_num_bytes(N);

 if ((cAB = OPENSSL_malloc(2*longN)) == NULL)
   return NULL;

 memset(cAB, 0, longN);

 EVP_MD_CTX_init(&ctxt);
 EVP_DigestInit_ex(&ctxt, EVP_sha1(), NULL);
 EVP_DigestUpdate(&ctxt, cAB + BN_bn2bin(A,cAB+longN), longN);
 EVP_DigestUpdate(&ctxt, cAB + BN_bn2bin(B,cAB+longN), longN);
```

- Fix - check relative size of fields:

```
  if (BN_ucmp(A, N) >= 0 || BN_ucmp(B, N) >= 0)
      return NULL;
```

Modern TLS Stack Memory Corruption

- Some exceptions:
  - gnutls session ID overflow: CVE-2014-3466
- From _gnutls_read_server_hello()

```
  /* Read session ID
 */
DECR_LEN(len, 1);
session_id_len = data[pos++];

if (len < session_id_len) {
  gnutls_assert();
  return GNUTLS_E_UNSUPPORTED_VERSION_PACKET;
}
DECR_LEN(len, session_id_len);
```

- Fix:

```
  if (len < session_id_len || session_id_len > TLS_MAX_SESSION_ID_SIZE) {
  gnutls_assert();
  return GNUTLS_E_UNSUPPORTED_VERSION_PACKET;
}
```

Integer Safety

- Protocol field integer size:
  - Typically UINT16, some UINT8, UINT24.
  - Most exposure to integer issues from ASN.1 lengths
  - Some exposure due to abnormally-small packet lengths
    - Block cipher / padding issues.

ASN.1 Integer Issue

- asn1_d2i_read_bio() overflow: CVE-2012-2110
    - Root cause - careless casting between ‘long’ and ‘int’
        - On 64-bit LP64 model systems, ‘long’ is 64-bit, ‘int’ is 32-bit.

ASN.1 Integer Issue

```
int ASN1_get_object(const unsigned char **pp, long *plength, int *ptag,
  int *pclass, long omax);
…
typedef struct asn1_const_ctx_st
  {
…
  long slen;  /* length of last 'get object' */
…
} ASN1_const_CTX;
…
#define HEADER_SIZE   8
static int asn1_d2i_read_bio(BIO *in, BUF_MEM **pb)
  {
…
  ASN1_const_CTX c;
…
  int want=HEADER_SIZE;
…
  int off=0;
…
  int len=0;
…
    p=(unsigned char *)&(b->data[off]);
    c.p=p;
    c.inf=ASN1_get_object(&(c.p),&(c.slen),&(c.tag),&(c.xclass),
      len-off);
…
```

ASN.1 Integer Issue

```
…
      /* suck in c.slen bytes of data */
      want=(int)c.slen;
      if (want > (len-off))
        {
        want-=(len-off);
        if (!BUF_MEM_grow_clean(b,len+want))
          {
 …
         goto err;
          }
        while (want > 0)
          {
          i=BIO_read(in,&(b->data[len]),want);
          if (i <= 0)
            {
…
            goto err;
            }
          len+=i;
          want -= i;
          }
        }
```

Small Record Bugs

- OpenSSL TLS CBc ciphers - explicit IV length greater than size of all blocks:
- Introduced by TLS explicit IV's needed for TLS 1.2 PRF changes.

```
--- t1_enc.c    2012/05/10 13:38:18    1.57.2.3.2.22
+++ t1_enc.c    2012/05/10 15:10:15    1.57.2.3.2.23
@@ -889,6 +889,8 @@
                if (s->version >= TLS1_1_VERSION
                        && EVP_CIPHER_CTX_mode(ds) == EVP_CIPH_CBC_MODE)
                        {
+                       if (bs > (int)rec->length)
+                               return -1;
                        rec->data += bs;    /* skip the explicit IV */
                        rec->input += bs;
                        rec->length -= bs;
```

Information Leakage

- Direct info leaks:
  - Heartbeat packets / OpenSSL heartbleed:
    - Unlikely to be repeated exactly.
    - Typical of info leaks in TLS.

Information Leakage

- Echoed fields occur elsewhere in protocol too.
    - Session ID.
    - Session tickets.
    - DTLS cookies.
    - Over OCSP:
        - OCSP status extensions (stapled responses).
        - OCSP response ID’s, and extensions.

Indirect Info Leaks

- Potential exists whenever parsing, interpreting, or validating data from out of bounds.
- Construct some type of oracle.
    - Out of bounds reads must be used in a discernible way:
        - Direct comparison of values, indirect usage for something interesting.
    - Most of these left in TLS stacks are in the handshake protocol.
        - Of limited utility.
            - HS protocol basically clear-text anyways, protected by finish message.

Secure Transport OOB Reads

- SSLDecodeDHClientKeyExchange()
- SSLDecodePSKClientKeyExchange()
- Server name extension parsing in SSLProcessClientHello()

Timing

- Indistinguishability of operations on plain text.
    - Mostly caused by padding.
        - An artifact of mac-then-encrypt in TLS.

Timing

- Constant-time operations:
  - CBC padding.
  - RSA master-secret decryption, padding checking.
    - Recommended: fill the pre-master secret with PRNG data.
    - PRNG's mix entropy with MD5.
  - MAC operations.

Timing Issues

- Can be influenced by design / language choices.
  - Botan: TLS stack written in C++.
- Uses C++ exceptions for error handling:

```
const size_t mac_size = cipherstate.mac_size();
const size_t iv_size = cipherstate.iv_size();

const size_t mac_pad_iv_size = mac_size + pad_size + iv_size;

if(record_len < mac_pad_iv_size)
   throw Decoding_Error("Record sent with invalid length");
```

Trust Errors

- Common causes:
  - Implementation errors in:
    - TLS protocol
    - PKI / X.509
  - Root of trust issues.

TLS Implentation Trust Errors

- ‘goto fail’ - fail to validate the signature on the encrypted pre-master-secret.

PKI / X.509 Trust Errors

- CN comparison truncation with NULL bytes (Kaminsky 2009).
- gnuTls ‘isCA’ bug (2014).
- NSS signature forgery bug (2014).
  - Failure to properly locate the hash in the decrypted signature.

Root of Trust Choice Issues

- Trusting the wrong roots:
  - Go Daddy Class 2 Certification Authority
    - Public exponent e = 3
- Of course, CA compromise, mistakes issuing certs, etc…

Concurrency Issues

- Design choices:
  - OpenSSL ‘SSL_SESSION *’ shared between multiple connections for multi-threaded servers.
  - Each session may be accessed concurrently on resumption, ’s->hit’ set if so:
    - Build state in session, then enter into cache, then never change.
    - Unless you forget to check ’s->hit’.

OpenSSL Session Concurrency Bugs

```
    if (type == TLSEXT_TYPE_ec_point_formats &&
                s->version != DTLS1_VERSION)
      {
      unsigned char *sdata = data;
      int ecpointformatlist_length = *(sdata++);

      if (ecpointformatlist_length != size - 1)
        {
        *al = TLS1_AD_DECODE_ERROR;
        return 0;
        }
      s->session->tlsext_ecpointformatlist_length = 0;
      if (s->session->tlsext_ecpointformatlist != NULL) OPENSSL_free(s->session->tlsext_ecpointformatlist);
      if ((s->session->tlsext_ecpointformatlist = OPENSSL_malloc(ecpointformatlist_length)) == NULL)
        {
        *al = TLS1_AD_INTERNAL_ERROR;
        return 0;
        }
      s->session->tlsext_ecpointformatlist_length = ecpointformatlist_length;
      memcpy(s->session->tlsext_ecpointformatlist, sdata, ecpointformatlist_length);
```

OpenSSL Session Concurrency Bugs

- Fix:

```
    if (!s->hit)
        {
        s->session->tlsext_ecpointformatlist_length = 0;
        if (s->session->tlsext_ecpointformatlist != NULL)
OPENSSL_free(s->session->tlsext_ecpointformatlist);
        if ((s->session->tlsext_ecpointformatlist =
OPENSSL_malloc(ecpointformatlist_length)) == NULL)
          {
          *al = TLS1_AD_INTERNAL_ERROR;
          return 0;
          }
        s->session->tlsext_ecpointformatlist_length =
ecpointformatlist_length;
        memcpy(s->session->tlsext_ecpointformatlist, sdata,
ecpointformatlist_length);
        }
```

Unexpected Attack Surface

- Authority Information Access extension in X.509v3 certificates - HTTP or LDAP client.

How They 'Stack' Up

- OpenSSL forks - BoringSSL wins over libressl.
  - Systematically checks input bounds on reads.
  - OpenSSL is an early adopter, and supports everything.
- All the OOB reads were pretty much still in libressl. and forks.

Comparing SSL Stacks

- General impressions:
  - OpenSSL forks - BoringSSL wins over libressl.
    - Systematically checks input bounds on reads.
  - All the OOB reads were pretty much still in libressl.

Comparing TLS Stacks

- Botan - C++ wins for bounds checking.
  - Concerns about constant-time and indistinguishability.
- PolarSSL - read without systematic checks, writes back to buffers.
  - Couldn’t find a bug.

- NSS - libpkix scares me, especially their AIA code.
- gnuTls - pretty uniform coding standards
  - Recent memory corruption bugs worry me, but I didn’t find anything.

Worst of the Worst

- The winner was: Secure Transport (OS X / iOS).
  - One feature away from being exposed, and shaky in so many ways.

Thanks!

- In my day job, I hunt for interesting malware, always looking for more samples.
- nmehta@google.com