Learning from SSL failures

@codecontext - BCN Cybersec meetup

DUAL EC DRBG

- Young&Yung paper: How to build cypto backdoors
- NSA EC DRBG Random number generator
- ANSI - Certicom, Entrust, Mastercard, NIST, NSA, RSA...
- ISO & NIST
  - One of 4 RNGs
  - Not the default algorithm
  - BUT x100 slower than others
- RSA's BSAFE - Default algorithm (10M$)
- FIPS 140-2 - Mandatory to pass the certification process

Notes

- Check what algorithms are your systems using (now)
- First check the experts community, then the standards

Debian random

- Use uninitialized `data + getpid()` as entropy source.
- Valgrind + Purify
- `/* MD_Update(&m,buf,j);  purify complains */`
- `srand(getppid())` ->32,768 seed values for deterministic private key generation
- Can pregenerate about 32k private keys

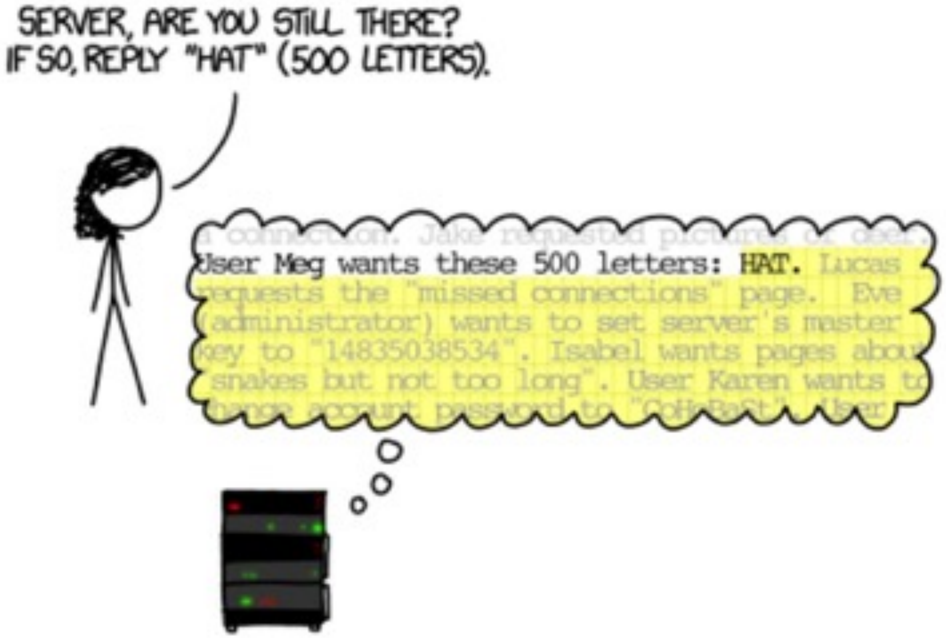
Notes

- Always use system source of entropy.
- Caution with code checkers
- Confusing comments?
- Check key space : srand(time(NULL)) , PIN, PUKs ...

HEARTBLEED

Copy **`in`** to **`out`**

```
/* Read type and payload length first */
hbtype = *in++;
n2s(in, payload);
out = OPENSSL_malloc(1 + 2 +
   payload + padding);
/* Enter response type, length and
   copy payload */
*out++ = TLS1_HB_RESPONSE;
s2n(payload, out);
memcpy(out, in, payload);
```



No malloc check?

No range checks?

Notes

- Force write preconditions and input validation in “unsafe” code
- Simplify the code to help code checkers to find bugs
- Software audits/reviews. Developers training.
- Use safer language for security development. C kills least privilege (Access to all memory).
- Negative unit tests for ill-formed L(ength)V(alue) packets. Deterministic fuzz testing.
- Always-zeroize-after-allocate-deallocate / store it in another subsystem (like HSM or Valut

Downgrade attack

SSLv2 Handshake messages are not protected. This permits a man-in-the- middle to trick the client into picking a weaker cipher suite than it would normally choose.

SSLv3 5.36.9. Finished Message

```
md5_hash:  MD5(master_secret + pad2 + MD5(handshake_messages + Sender
  + master_secret + pad1));
sha_hash:  SHA(master_secret + pad2 + SHA(handshake_messages + Sender
      + master_secret + pad1));
```

TLSv1.2 7.4.9 Finished Message

```
   PRF_SHA256(master_secret, finished_label, SHA256(handshake_messages))
   [PRF=Pseudo random function]
```

Notes

- All data sent in the wire should be authenticated before processing the message

goto fail

```
if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
  goto fail;
  goto fail;

if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
        goto fail;
        ...

fail:
        SSLFreeBuffer(&signedHashes);
        SSLFreeBuffer(&hashCtx);
        return err;
```

Clang has -Wunreachable-code to warn about this, but it's not in -Wall.

Notes

- Understand what kind of checks are doing your software checkers

CRIME (Compression ratio info-leak make easy)

SSLv2 Data are not compressed before encryption. Compression eliminates most structure and redundancy from the plaintext.

```
SSLv3 5.2.2. Record Compression and Decompression
All records are compressed using the compression algorithm defined in the
current session state. There is always an active compression algorithm;
however, initially it is defined as CompressionMethod.null.
```

If session cookie `session=1929` TLS record is smaller than if cookie `session=2929`.

```
<html><body>
   <img src="https://bank.com/foo1.jpg?session=1>
   <img src="https://bank.com/foo2.jpg?session=2>
```

Notes

- Remove-redundancy-before-cypher is not ALWAYS a good practice
- Length of ciphered messages could give information to attackers.
- Optimizations could give side-channel information.
- Attacker has access to “cipher engine”

FREAK

- cli>srv: Use RSA2048
  - MITM: cli>srv: Use RSA512
- srv>cli: Ok, We’ll use cipher RSA512
  - Client also does not check RSA2048! =RSA512
- cli <-> srv: ciphered communications
  - MITM: record all ciphered communications
- MITM: Factor RSA512 (100$ AWS)
- MITM: Decipher communications

Notes

- Check if negotiation results matches with your initial requirements

LOGJAM

- MITM: Downgrade to DH_EXPORT
- DH_EXPORT uses a massive reused public known key by default

	Vulnerable if most common 1024-bit group is broken
HTTPS – Top 1 Million Domains	17.9%
HTTPS – Browser Trusted Sites	6.6%
SSH – IPv4 Address Space	25.7%
IKEv1 (IPsec VPNs) – IPv4 Address Space	66.1%

Notes

- If you are reusing a key, study the pros/cons
- Remove support of insecure crypto algorithms from code

Key Exchange Algorithm Confusion

```
struct {
  /* KeyExchangeAlgorithm algorithm */
  select (KeyExchangeAlgorithm) {
    case diffie_hellman:
        ServerDHParams params;
        Signature signed_params;
    case rsa:
        ServerRSAParams params;
        Signature signed_params;
   }
} ServerKeyExchange;
```

Theoric attack. Algorithm is context-bounded. Attacker could drop/alter some protocol messages to change the meaning of the messages

Notes

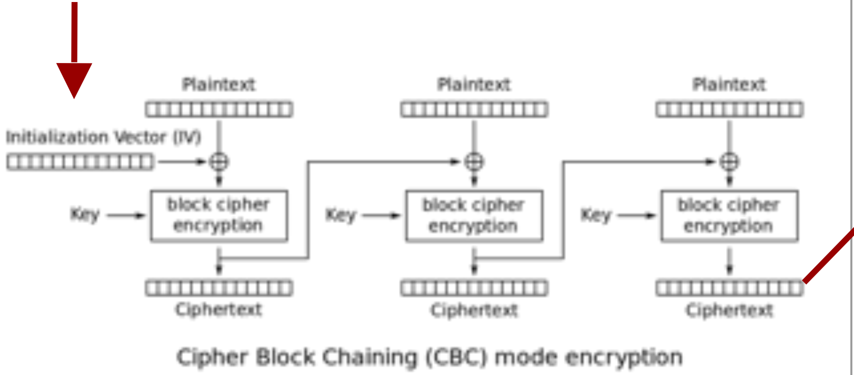
- Use context-free message structures: Misinterpretation of a received message should be avoided by providing explicit information on the content.
- Tagging each field with some information indicating its intended type.
- Version “meanings” of data

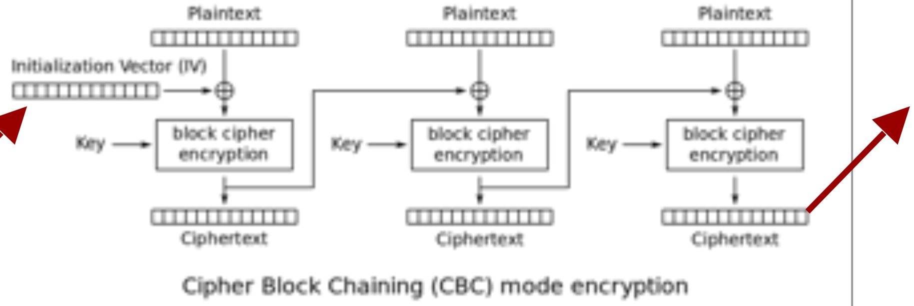
BEAST

TLSv1

initialization vector (IV) for the first record is generated with the other keys and secrets when the security parameters are set. The IV for subsequent records is the last ciphertext block from the previous record.

func(session_data)

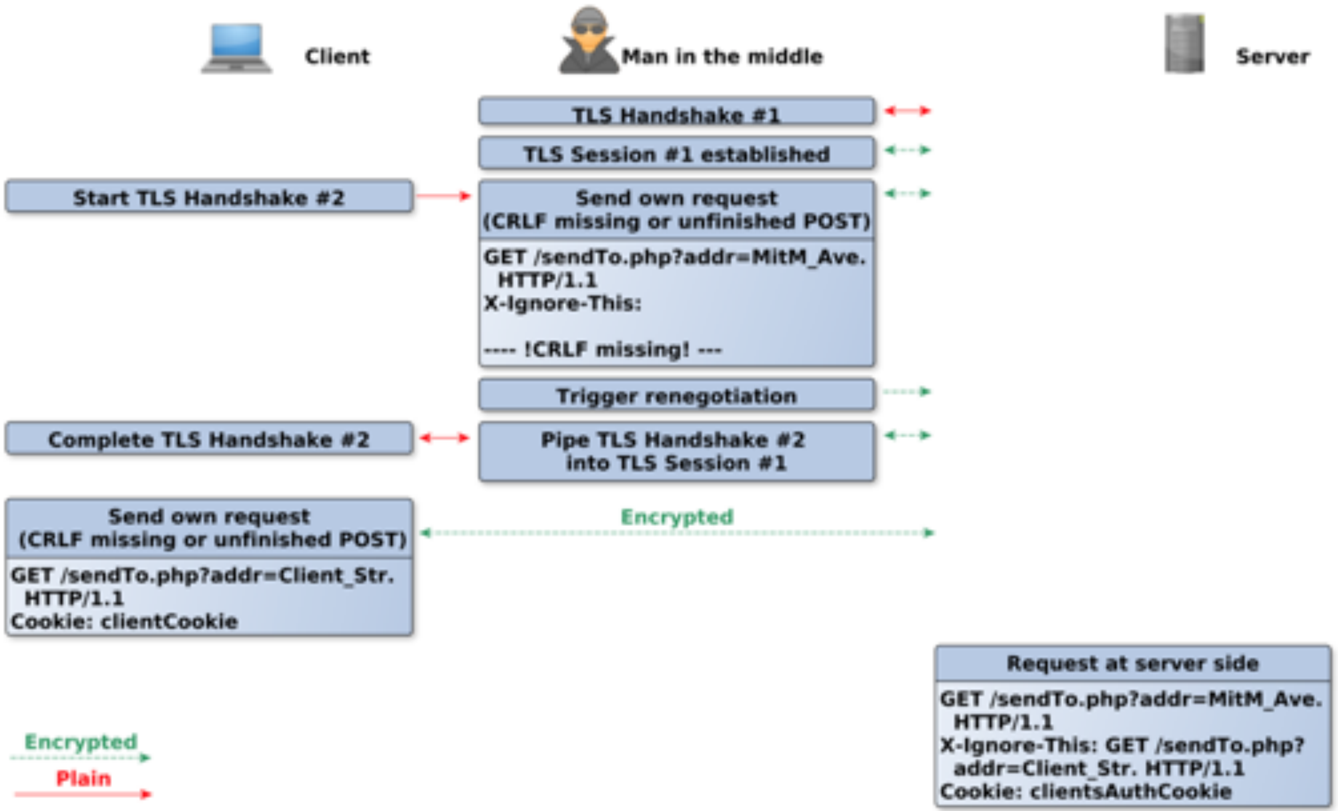




Notes

- Use random IVs / don’t reuse any IV data
- Entropy needed in each independent network packet
- Discard CBC and use AEAD like CGM.
- A crypto algorithm is bounded to a context

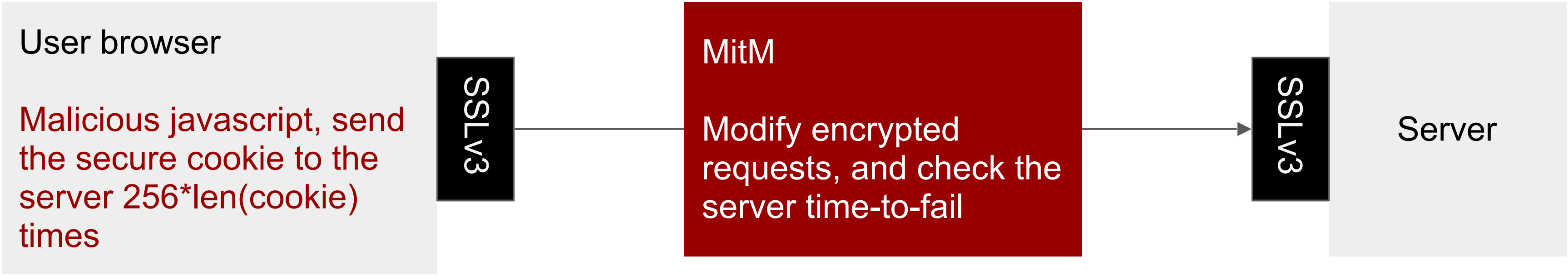
Renegotation attack



Notes

- When switching security contexts it needs to be guaranteed that there is no pending data left

POODLE


User browser
Malicious javascript, send the secure cookie to the server 256*len(cookie) times
SSLv3
MitM
Modify encrypted requests, and check the server time-to-fail
SSLv3
Server

POODLE

```
msg = cukie
Message Authentication Digest of msg= MAC
```

```
block1                          block2
------------------------        ------------------------
M  A  C  c  u  k  i  e          .  .  .  .  .  .  .  7
```

Padding content is **not specified** (in this case 0’s)

```
M  A  C  c  u  k  i  e          0  0  0  0  0  0  0  7
          ||                               ||
          \/                               \/
iv ->   ( xor )          /-----------> ( xor )
          ||             |                 ||
          \/             |                 \/
key ->  (cipher)         |      key-> (cipher)
          ||             |                 ||
          \/             |                 \/
C0 C1 C2 C3 C4 C5 C6 C7 -/      C8 C9 Ca Cb Cc Cd Ce Cf
```

POODLE

```
SERVER
============
1 - Read blocks
2 - Check padding
3 - Check MAC
```

```
C0 C1 C2 C3 C4 C5 C6 C7  -\      C8 C9 Ca Cb Cc Cd Ce Cf
          ||              |                ||
          \/              |                \/
key -> (decipher)         |       key-> (decipher)
          ||              |                ||
          \/              |                \/
iv ->   ( xor )           \----------->  ( xor )
          ||                               ||
          \/                               \/
M  A  C  c  u  k  i  e           .  .  .  .  .  .  .  7
```

Server only checks last byte

1 verification then FAIL (1s) => PADDING ERROR (message size is in upper network layer)
2 verifications then FAIL (2s) => MAC ERROR
2 verifications then SUCCESS (2s) => OK

POODLE

Valid decrypted plaintexts

M	A	C	c	u	k	i	e	0	0	0	0	0	0	0	7
M	A	C	c	u	k	i	e	0	0	0	0	0	0	1	7
M	A	C	c	u	k	i	e	0	0	0	0	0	0	2	7
M	A	C	c	u	k	i	e	0	0	0	0	0	0	3	7
M	A	C	c	u	k	i	e	0	0	0	0	0	0	4	7
M	A	C	c	u	k	i	e	0	0	0	0	0	0	5	7
M	A	C	c	u	k	i	e	0	0	0	0	0	0	6	7
M	A	C	c	u	k	i	e	0	0	0	0	0	0	7	7
.	.	.													
M	A	C	c	u	k	i	e	F	F	F	F	F	F	F	7

0xFFFFFFF valid plaintexts. If MitM sends a random 2nd packet and there's a MAC error then we know that plaintext of random packet the finishes with '7'

POODLE

- POODLE attack is solved in TLS v1 filling all the padding (PKCS#7)

```
block1                                  block2
------------------------                ----------------------
M   A   C   c   u   k   i   e           7   7   7   7   7   7   7   7
```

- TLSv1: Does not differenciate between MAC and padding errors. (Handling of padding errors changed from bad-record-mac rather than decryption-failed alert to protect against CBC attacks)
- TLSv1.2 … In general, the best way to do this is to compute ... **This leaves a small timing channel since ..., but it is not believed to**… **LUCKY13 on LAN!**
- POODLE still valid in some TLS implementation (F5, A10, IBM, Cisco, ...) because are using SSLv3 padding verification algorithm

Notes

- All data shall mean something.
- Encrypt the plaintext, then append a MAC of the ciphertext.
- Consider constant timing verification functions in each layer
- Do not send detailed error information (server shouldn’t be an Oracle). One bit disclosed x attack (oracle) and attack can be repeated 16k times => 16k disclosed bits
- An attacker that could cipher data with your keys (without knowing them) shouldn't break your ciphered data.
- Be careful duplicating data (padding size in message + message length in other layer). Enigma

Notes

```
int CRYPTO_memcmp(const volatile void * volatile in_a,
                  const volatile void * volatile in_b,
                  size_t len) {
   size_t i;
   const volatile unsigned char *a = in_a;
   const volatile unsigned char *b = in_b;
   unsigned char x = 0;
   for (i = 0; i < len; i++)
       x |= a[i] ^ b[i];
   return x;
}
```

https://golang.org/src/crypto/subtle/constant_time.go

https://cryptocoding.net/index.php/Cryptography_Coding_Standard

STATE MACHINE

- SMACK
  - OpenSSL
  - GnuTLS
  - NSS
  - Java
  - Mono
  - CyaSSL

Notes

- Understand, draw the full state machine
  - It could easy become a very messy diagram
  - Including sub-protocols: timeouts & errors in upper & lower transports
  - *Input-enabled* state machines: So for every state the state machine shall specify what should happen for every possible input.
- Implement it ..or in *happy flow* legacy code track the state machines changes and define well-known trusted paths using the current usage

?

Gràcies

adriamassanet (at) gmail.com
@codecontext

Some references

https://eprint.iacr.org/2013/049.pdf
http://www.thoughtcrime.org/blog/the-cryptographic-doom-principle/
https://en.wikipedia.org/wiki/Authenticated_encryption
https://www.imperialviolet.org/2014/02/22/applebug.html
https://www.ietf.org/proceedings/85/slides/slides-85-saag-1.pdf
http://faratarjome.ir/u/media/shopping_files/store-EN-1426494440-2731.pdf
https://www.schneier.com/blog/archives/2008/05/random_number_b.html
https://www.cs.bham.ac.uk/~mdr/teaching/modules06/netsec/lectures/tls/tls.html
http://blog.cryptographyengineering.com/2015/01/hopefully-last-post-ill-ever-write-on.html
http://www.dwheeler.com/essays/heartbleed.html
https://www.cs.ru.nl/E.Poll/papers/langsec.pdf
http://www.ieee-security.org/TC/SP2015/papers-archived/6949a535.pdf
https://mitls.org/pages/attacks/SMACK
http://www.yaksman.org/~lweith/ssl.pdf
https://www.schneier.com/cryptography/paperfiles/paper-ssl-revised.pdf
http://www.hackmageddon.com/2011/09/25/the-beauty-rc4-and-the-beast-tls/