ASN.1 security issues

Octet-based and character-based
transfer syntaxes security vulnerabilities
ASN.1 secured decoders implementation guidelines

Version 0.0.8

http://www.powerasn.com

Octet-based encoding example

Here is a DER encoding of the following definition:

```
Person ::= SEQUENCE {
        first   UTF8String,
        last    UTF8String
}
```

```
myself ::= Person {
        first "Nathanael",
        last "COTTIN"
}
```

30 13
0C 09 4E 61 74 68 61 6E 61 65 6C
0C 06 43 4F 54 54 49 4E

What’s wrong with it?

ASN.1 vulnerabilities

My

other computer

is

YOURS!

Security issues

Both TLV-based and character-based encodings may transport malicious extra (and potentially dangerous) information

Many possible attacks when improper decoders design

Guidelines for secured decoders implementation

Octet-based encodings

ASN.1 defines 3 octet-based encoding rules:

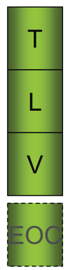
- BER
- CER
- DER

TLV-based

PER is not discussed here (bit-based encoding)

Focus on TLV-based transfer syntaxes

TLV encoding remainder



T = Type or tag

L = Length (definite or indefinite)

V = Value (L octets if definite-length)

Optional "End-Of-Content" octets if indefinite-length

V = TLV when EXPLICIT tagging or constructed form

Please refer to encoding rules presentation

Possible implementation weaknesses

Infinite streams

Missing "End-Of-Content" (EOC) octets

Truncated encoding

Negative length

Value overrun

Denial of Service

Buffer overflow (see appendix for details)

Extra data injections

Malicious codes execution

Infinite streams

Problem:

Decoder receives an indefinite-length BER-encoded stream. The sender loops until the receiver crashes (out of memory, buffer overflow, etc.)

Solution:

Set a maximum length and raise an error if received length exceeds this limit

Drawback:

The destinatory ASN.1 object may not be initialized even if the received encoding is valid (but too large)

Missing “End-Of-Content” octets

Problem:

Decoder receives an indefinite-length BER-encoded stream. The sender does not send the “End-Of-Content” octets and does not close the stream. The receiver waits indefinitely…

Solution:

Set a maximum timeout

Drawback:

The destinatory ASN.1 object may not be initialized depending on the network load (timeout reached)

Truncated encoding

Problem:

Decoding can be made incomplete when EOC injected within received encodings (even during transmission…)

The objective is to make the decoder think it has reached the end of the encoded data before the actual end of the stream. The attacker may even send an end-of-stream flag

Solution:

Concerns indefinite-length encodings

(Definite-length needs to modify octets in cascade and thus is more difficult to implement)

Negative length (alternative 1)

Problem:

Decoder receives a octet-based encoding with a negative definite length. The decoder may crash during the corresponding memory allocation

Solution:

Check that the received length is positive before allocating memory

Negative length (alternative 2)

Problem:

Decoder receives a long definite-length octet-based encoding. The encoding is well-formed but decoded length is too large and does not fit into the “length” variable (integer overrun). Negative length error may also apply

Solution:

Check that received length is positive (length becomes negative if too large)

Drawback:

The destinatory ASN.1 object may not be initialized even if the received encoding is valid (but length is too large for the decoder implementation)

Value overrun

Problem:

Decoder receives a valid octet-based encoding. The length marker can be successfully decoded. However, the (partly) decoded value does not fit into the corresponding type implementation

Solution:

Check that received length of the encoded value is lower than a maximum size limit

Drawback:

The destinatory ASN.1 object may not be initialized even if the received encoding is valid (but encoded value is too large for the ASN.1 object value implementation)

Buffer overflow

Problem:

Decoder receives an octet-based encoding. The encoding is well-formed but very large. The decoder may crash (overflow error) if the temporary buffer is not able to receive the whole encoding (size < received data length)

Solution:

Use streams or lists (i.e. linked lists) instead of arrays

Combine with maximum decoding size limit

Reinitialize all buffer to prevent code from being executed

Extra data injections

Extra data can be injected within TLV-based encodings in different locations:

- Appended at the end of the stream (primitive or constructed encoding)
- Appended at the end of an EXPLICIT encoding UNIVERSAL encoding
- Appended at the end of a constructed encoding

↳ Rely on a length-value incoherence or indefinite-length checks leaks

Extra data injection overview

Overview of 2 common TLV-based injection cases:

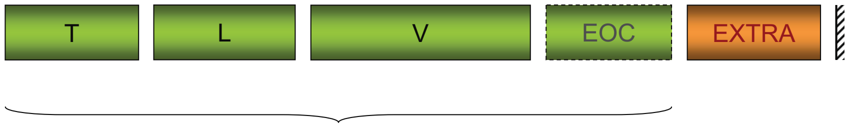
- Definite-length encoding:

T	L (definite)	V (L octets)	EXTRA

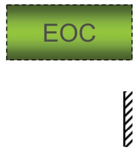
- Indefinite-length encoding:

T	L (indefinite)	V	EXTRA	EOC

Before end-of-stream injection



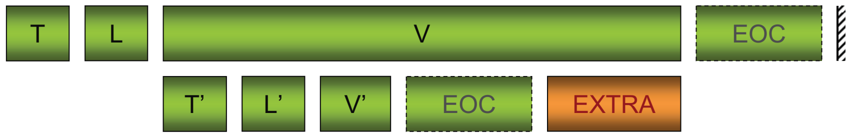
Valid definite-length or indefinite-length, primitive or constructed main encoding



= Optional “End-Of-Content”

= End of stream

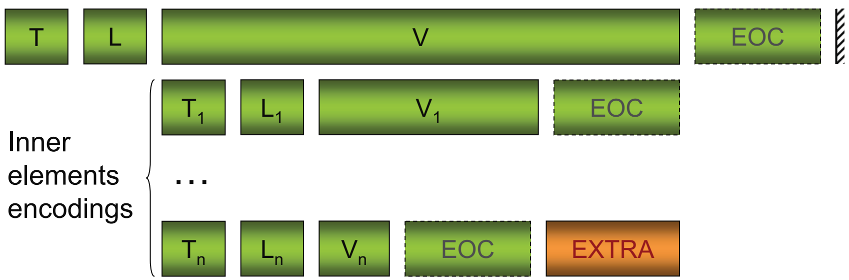
Inside an EXPLICIT encoding



- Definite-length only: L and L’ definite format
  - Check that L’ = L – length of T’ encoding
- Mixed length: L definite and L’ indefinite format
  - Construct L’ actual length (definite-form) when reading V’ and check against L

End of a constructed encoding

EXPLICIT encoding flow generalization:



Extra information injected after last inner element encoding

Character-based encodings

ASN.1 defines an XML-based transfer syntax (XER and its variants)

- XML decoding security issues apply
- ASN.1 XER specific encodings flows added

General XML security issues – 1/3

- Infinite streams (unbounded elts)
- Missing end tag: </ . . .>

Similar to octet-based decoding issues

- XML comments used to:
  - Inject code in RAM
  - Generate non-terminated or truncated encodings
  - Replace values
- Extra tags insertion: not possible if decoding validated using a DTD
- Use of external DTDs and XMLSchemas: possible security holes during the XML validation process (and infinite loops: DTDs circular references)

General XML security issues – 2/3

Example : use comments to generate a non-terminated encoding

my-first ::= first UTF8String “Nathanael<!--”

XER encoding is:

<first>Nathanael<!--</first>

This is a non-terminated encoding!

General XML security issues – 3/3

Example : String invalid encoding

Value may mention beginning or closing XML tags, cheating decoders about the end of the received data

If encoded string is “Natha</first>nael”, then XML encoding is:

What happens to this part?

<first>Natha</first>nael</first>

XER-specific security issues

Example : OCTET STRING types encoding use base64 to encode values

Base64 (PEM) encoding weaknesses: extra data insertion at the end of the last base64 or PEM line:

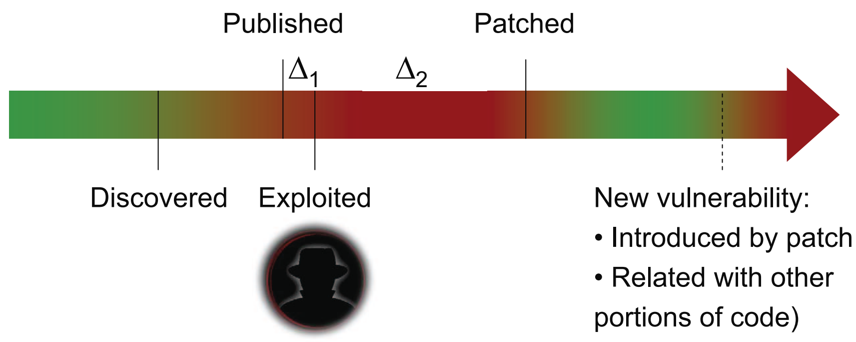
- After the equal -“=“- sign if padded
- After the last character -64th character- if no padding

Other common issues

Non related with a particular transfer syntax:

- Use of external libraries / programs:
  - Regular expressions matchers to validate strings
  - Invalid two's-complement representations of integers
  - XML encoders and decoders: may not fit with XER
- Design issues:
  - Let the user more freedom than necessary
  - Incomplete understanding of the program to create

Reminder: vulnerability lifecycle



Generally $\Delta_1 << \Delta_2$

Conclusion

Take care of security issues when designing decoders

Security must be part of the decoders specifications and not a patch…

Use carefully third-parties material to avoid “external” weaknesses

> The more widely used, the more attention paid on security implementations gaps

Appendix A: buffer overflow principle

- Function call: EIP pointer pushed in stack
- Function execution: allocates data on stack (params)
- End of function call: EIP popped to continue main program execution

Buffer overflow attack: use more space than allocated
→ EIP address overwritten on stack

End of call followed by injected code execution

Appendix B: Microsoft vulnerabilities

Microsoft ASN.1 Library Length Overflow Heap Corruption

Release Date:
February 10, 2004

Severity:
High (Remote Code Execution)

Systems Affected:
Microsoft Windows NT 4.0 (all versions)
Microsoft Windows 2000 (SP3 and earlier)
Microsoft Windows XP (all versions)

Software Affected:
Microsoft Internet Explorer
Microsoft Outlook
Microsoft Outlook Express
Third-party applications that use certificates

Services Affected:
Kerberos (UDP/88)
Microsoft IIS using SSL
NTLMv2 authentication (TCP/135, 139, 445)

BER decoding error in MSASN1.DLL library. Affects Microsoft crypto API. Concerns lengths from 0xFFFFFFFD to 0xFFFFFFFF

Appendix C: OpenSSL vulnerabilities

OpenSSL ASN.1 DoS Vulnerabilities

By Dr. S. N. Henson

1. During the parsing of certain invalid ASN.1 structures an error condition is mishandled. This can result in an infinite loop which consumes system memory (CVE-2006-2937). This issue did not affect OpenSSL versions prior to 0.9.7

2. Certain types of public key can take disproportionate amounts of time to process. This could be used by an attacker in a denial of service attack (CVE-2006-2940)

Any code which uses OpenSSL to parse ASN.1 data from untrusted sources is affected. This includes SSL servers which enable client authentication and S/MIME applications