Title: **Certifying/verifying software compilation as a solution to ASN.1 implementations vulnerabilities**

SUMMARY

CERT/CC identified vulnerabilities in SNMP and OpenSSL implementations related to the use of ASN.1. Ill-formed ASN.1 Tag/Length/Value (TLV) structures and weak parsers and decoders are responsible for most of these vulnerabilities and not the ASN.1 language itself. As a consequence, exploiting SNMP and OpenSSL vulnerabilities in an operating environment may lead to failures and denials of services. ASN.1 is used in a number of important protocols; therefore it is very important to ensure that ASN.1 implementations are safe. This contribution is presented for information sharing and to generate interest in the so-called *language-based approach to security and certifying compilation techniques* to ensure that a protocol implementation is safe before it is deployed in an operating environment.

1 Introduction

This contribution is presented for information sharing and to generate interest in the subject it addresses. Abstract Syntax Notation One (ASN.1) is used in several important protocols deployed in various environments. CERT/CC identified a number of vulnerabilities (weaknesses) in implementations of the Simple Network management Protocol (SNMP) and Open Secure Socket Layer (OpenSSL) related to the use of ASN.1. The vulnerabilities found can lead to threats and attacks against an organization's computing and networking assets. In the testing environment of SNMP, they led to buffer overflow conditions, equipment failures and denials of services.

One of the main objectives of this contribution is to identify the possible causes of the vulnerabilities found in SNMP and OpenSSL implementations and to suggest solutions. Another objective is to discuss the use of the so-called *language-based approach to security techniques and certifying compilers* to ensure that a protocol implementation is safe before it is deployed in an operating environment.

The contribution is organized as follows: Section 2 describes SNMP testing done at Oulu University, the weaknesses found in SNMP implementations and their impacts. Section 3 describes OpenSSL vulnerabilities. Section 4 discusses the causes of the vulnerabilities of ASN.1 implementations. Section 5 is an overview of language-based security techniques, certifying compilers and their applicability to protocol implementations. Finally, section 6 is the conclusion.

Note on the terminology: The words "certifying", "certified", "certification", "certificate" and all other variations are used in this contribution as they are defined in the field of programming language theory and compiler design to refer to a compiler that produces evidence that the code it produce is correct and safe to execute. A *certifying compiler* is a compiler that not only produces target code but also a certificate, a machine-checkable evidence that the code respects safety policies. In ITU-T SANCHO database, the word "certification" is assigned a different meaning than the one used in this contribution. With this clarification, no ambiguity should occur.

2 Testing SNMP Implementations at Oulu University

The Oulu University Secure Programming Group (OUSPG) in Finland has reported numerous vulnerabilities in SNMP v1 [RFC1067] in 2002 after stress testing a number of implementations from many vendors [PROTOS]. OUSPG results were documented in two CERT/CC vulnerabilities notes [CERT1, CERT2]. The OUSPG revealed vulnerabilities in the way many SNMP

Contact: Claude Kawa, John Mullins
École Polytechnique Montréal
Robert Charpentier
DRDC Valcartier

Email: claude.kawa@sympatico.ca
John.Mullins@polymtl.ca
Robert.Charpentier@drdc-rddc.gc.ca

implementations decode and process SNMP messages. Vulnerabilities in the decoding and subsequent processing of SNMP messages may result in denial-of-service conditions and buffer overflows.

2.1 Coverage of SNMP tests

OUSPG tested several implementations from different vendors of SNMP v1 and TCP/IP Management Information Base (MIB) defined in RFC 1156 [RFC1156]. Most of them failed at least one of the tests [PROTOS]. The purpose of the tests was to assess the behaviour and reaction of an SNMP receiver when it is subjected to SNMP messages containing syntactic errors and unusual and extreme values in the different components of a TLV. The tests were exercising mainly SNMP TLV decoders and some of them tested the possibility of buffer overflow.

The tests covered the following areas:

- Invalid bit patterns in TLV (test series B)
- BER encoding errors (test series E)Exceptional string formats (test series F)Exceptional integer values (test series I)Missing symbol exceptions (test series M)Overflow conditions (test series O)**Overview of test series B:** This series of tests covered error codes returned by a receiver to a sender to indicate the type of error found in the SNMP message received.

Overview of test series E: This series of tests covered various cases of TLV encoding. Tests cases for the tag, length and value fields were performed with invalid values. The purpose of this series of tests was to find out how robust was the SNMP decoder of the implementation under test (IUT).

Overview of test series F: This series of tests checked the decoding by the IUT of unusual string formats.

Overview of test series I: This series of tests covered unusual integer values: Boundary values (integers close to the minimum and maximum integers) and overflow values (very large or very small integers) that could cause an overflow condition in the IUT.

Overview of test series M: The purpose of the tests of this series was to check the reaction of the IUT with a zero-length value.

Overview of test series O: This series of tests covered several cases of overflow conditions. For example, very long strings of characters were sent to the IUT.

2.2 Consequences of SNMP Vulnerabilities

OUSPG tests found multiple vulnerabilities or weaknesses in the way many SNMP managers decode and process Trap messages sent by agents [CERT2] and multiple vulnerabilities in the way request messages are decoded and processed by an agent [CERT1]. Request messages are sent by managers to agents to either request information or to instruct an agent to configure a managed equipment.

In an operating or production environment, an SNMP implementation could encounter a condition similar to that of the tests and fails. The tests showed different types of failures:

- A fatal failure that will cause the equipment to stopA crash of SNMP implementation that will require a manual restart of the equipment
- A crash of SNMP implementation followed by an automatic restart
- A failure of SNMP that will cause a freeze of the equipment or a consumption of the entire available memory

When an SNMP implementation encounters one of the above failures, it will result in a denial of service situation since it will not be able to respond to legitimate requests until it recovers from the failure. Furthermore, a malicious user could exploit SNMP implementation vulnerabilities to intentionally cause harm.

2.3 Possible explanations of SNMP lack of robustness

Several reasons can explain SNMP implementations lack of robustness:

1. RFC 1155 that describes SNMP MIB syntax lacks rigor and clarity. Therefore it is possible that SNMP tool designers misinterpret some RFC 1155 and ASN.1 Recommendation statements and produce incorrect parsers, coders or decoders.
2. Several SNMP implementations use third-party ASN.1 library functions to encode and decode SNMP messages. Third party encoding and decoding functions are generic. They check SNMP messages conformance to RFC 1155 and 1156 and not to a specific MIB with possibly tight sub-types. For example, a MIB may define a range for an integer object. If a transmitter encodes values outside the range, they will be undetected by an ASN.1 decoder and if SNMP agent or manager subsequently does not perform additional checking, invalid values will be accepted.
3. Some MIBs do not use subtypes (for example they will use the type "integer" instead of a small range). If subtypes are not used, a transmitter could encode unexpected values that will potentially cause a receiver to crash if its decoder does not check between legal and illegal values or does not expect exceptional values.
4. In many cases the same vendor supplies both SNMP agents and managers. As long as they interact with one another, they will work fine however if they have to interact with a third party implementation, they may fail or cause the other side to fail.

3 OpenSSL vulnerability

3.1 SSL and ASN.1

The Secure Sockets Layer (SSL), originally developed by Netscape Communications, is a popular protocol that allows a browser to securely access a Web server. The main role of SSL is to provide security for web traffic. SSL security functions include confidentiality, message integrity, and authentication. SSL realizes these functions through the use of cryptography, digital signatures, and certificates.

To authenticate servers and optionally clients, SSL uses X.509 certificates to validate identities [X509]. X.509 certificates contain information about the entity (for example a web server), including its public key and name. The certificate is signed by a certificate authority to give confidence to the certificate receiver about the validity of the sender. X.509 Certificates are specified in ASN.1.

3.2 Nature and Impact of SSL vulnerability

According to CERT Vulnerability note VU #748355 [CERT3], the ASN.1 library used by OpenSSL [OSSL] has several parsing errors that allow invalid certificate encodings to be parsed incorrectly and considered valid. More specifically, the parser implemented in various ASN.1 libraries accepts certain invalid TLV length field leading to improper interpretation of the data. Affected ASN.1 library functions include those supporting SSL and TLS protocols.

4 Vulnerabilities of ASN.1 implementations

4.1 Causes of vulnerabilities

There are four possible causes to ASN.1 implementation vulnerabilities:

- Protocol designers use generic ASN.1 data types (e.g. integer instead of a range) enabling an implementation to send unexpected large values causing its peer to crash,
- ASN.1 compiler translates an ASN.1 subtype to a more generic programming language type (e.g. an integer range to a integer) allowing implementations to send invalid values to the other end,
- ASN.1 coder generates incorrect ASN.1 TLV (with errors in tag, length or value fields),
- ASN.1 decoder generates incorrect data structures from ASN.1 transfer syntax byte streams.

4.2 Remedies against vulnerabilities

What are the remedies against ASN.1 implementation vulnerabilities? There could be three types of remedies:

- **Against sloppy protocol design:** Enforce the use of sub-typing. One way is to use an ASN.1 editor that prevents the use of ASN.1 generic types such as integer. This is difficult to enforce since most ASN.1 specifications are written in standard bodies and for some applications unusual data types are required, for example very long integers. Another possibility is to write type safety policies in a declarative language to override the generic types used in an ASN.1 specification.
- **Against ASN.1 compilers:** Ensure that the most appropriate mapping is done between ASN.1 types and the corresponding types of the programming language. There are cases with some commercial ASN.1 compilers where an ASN.1 sub-type is converted to a generic Java type.
- **Against ASN.1 coders and decoders:** This is a matter of ensuring the correct and robust implementation of the coder and the decoder; obviously it is not a simple matter. The next section discusses this item in some details.

4.3 Remedies against faulty coders and decoders

4.3.1 Classification of TLV errors

TLV errors are either caused by a faulty coder or a faulty decoder or both.

Errors caused by a faulty coder:

- Encoding of invalid tag value,
- Encoding of an invalid length inconsistent with the actual length of the value field,
- Invalid contents of the value field inconsistent with the tag,
- Abnormal long TLV.

Errors caused by a faulty decoder:

- Unable to cope with an invalid TLV,
- Acceptance of an incorrect TLV that will cause problems later to the hosting computer.

4.3.2 Prevention of TLV errors:

Against errors caused by a faulty coder:

- Ensure correct tag values are used,
- Ensure length field is consistent with the actual length of the value field,
- Ensure contents of the value field is consistent with the tag,
- Ensure that there are no abnormally long TLV.

Against errors caused by a faulty decoder:

- Ensure correct parsing and decoding of well-formed TLV,
- Ensure a robust implementation of the decoder to cope against a faulty or malicious coder.

4.3.3 Protocol conformance:

If we look at the problem in terms of protocol conformance, for the coder we need to ensure that:

- Its output (transfer syntax) is equivalent to the abstract syntax,
- It does not generate anything that is not in the abstract syntax.

For the decoder we need to ensure that:

- Its output (concrete syntax) is equivalent to the abstract syntax corresponding to the byte stream (transfer syntax) received and decoded,
- It does not generate anything that is not in the abstract syntax

Things get complicated because the coder and decoder do not deal directly with the abstract syntax (see Figure 4.1) but only with the transfer syntax and concrete syntax. So we have to assume that the concrete syntax is correct and acts as a surrogate for the abstract syntax.

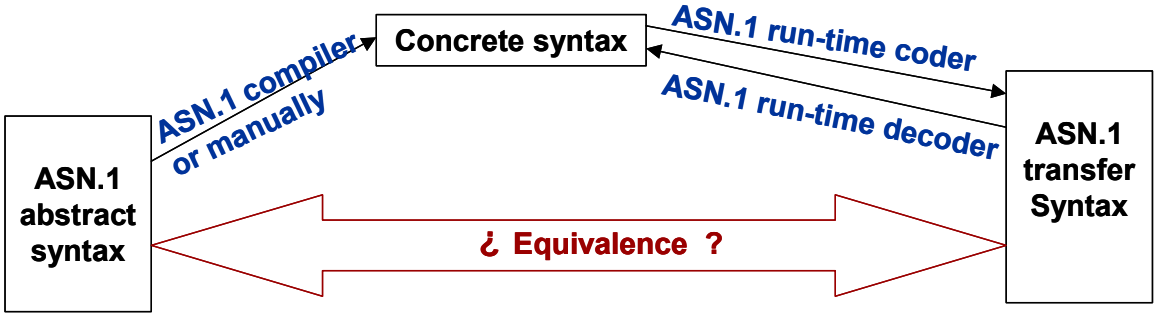

Figure 4.1 – Equivalence between the abstract and concrete syntax

However, one important question is whether we can rely on ASN.1 compilers to generate:

1) The correct data structures in the concrete syntax corresponding to ASN.1 data types expressed in the abstract syntax,
2) The corresponding correct coders and decoders.

The answer is no. We cannot rely on ASN.1 compilers to generate the correct data structures with the correct coders and decoders because such compilers can be faulty in several ways. So basically we have two problems:

1. The robustness of ASN.1 decoders to cope with incorrect TLV and the correctness of the coders and decoders. By correctness we mean that assuming the input of a coder (or decoder) has a certain value, the output of the coder (or decoder) must have a correct value depending on its inputs.
2. The correctness of ASN.1 compiler to generate the correct concrete syntax corresponding to the abstract syntax of the ASN.1 specification being compiled. This

problem is about showing the correctness of an ASN.1 compiler and conformance to a protocol specification.

We do not deal directly with this second problem is this contribution.

Solving part of the first problem will ensure that**:**

- Correct tag values are used,
- The length field is consistent with the actual length of the value field,
- Ensure contents of the value field is consistent with the Tag,
- Ensure that there are no abnormal long TLV,
- Ensure correct parsing and decoding of well-formed TLV.

However, we will not ensure that the compiler generates the correct data types and structures (second problem) nor that the decoder is robust to cope against a malicious or faulty coder, these are two separate questions.

5 Language-based approaches to software safety

5.1 Overview of certifying compilation

The growing use of mobile code with web applications has created a research trend since the nineties to address the issue of protection against malicious code using the so-called *language-based approach to security* [McM00]. Language-based approach to security is a set of techniques based on programming language theory and implementation, program analysis (including type checking), abstract interpretation, proof checking, program rewriting and program verification to ensure that the code is safe to execute [SMH, Ko99].

Certified compilation is a technique requiring that a compiler produce evidence that the code is safe to execute. A *certifying compiler* is a compiler that not only produces object code but also a certificate, a machine-checkable evidence that the object code respects safety policies such as:

1- Instruction safety (restriction on instruction the component can execute),

2- Memory and type safety (the program does not access unauthorized memory locations and the variables are of the right types),

3- System-call safety (restrictions on what system functions the component can invoke) and,

4- Partial correctness (restriction on the input/output behaviour of a component entry/exit points.

Such compilers were built for a type-safe subset of the C programming language [Ne98, NL98] and for Java [CLB00]. However, these projects are far from complete.

5.2 Certified compilation and Programming by contract

Programming by contract is a metaphor used for the first time by Bertrand Meyer [Me88] to describe the relationship between a class and its clients. Programming by contract sees this relationship as a formal agreement expressing each party's rights and obligations. At the heart of programming by contract are the pre- and post-conditions attached to a function specifying the contract between the client and the function (Figure 5.1). The pre-condition states that if the client calls the function with input parameters satisfying the pre-condition, in return the function promises to deliver the results satisfying the post-condition.

The use of pre- and post-conditions is defined, for example, for the programming language Eiffel [Me92]. The capability to include assertions in a program was added to the Java programming language [Jv] but it is more rudimentary than in Eiffel.

Is there any use to the assertion capabilities found in Eiffel and Java for the ASN.1 problem? We could write ASN.1 coder and decoder in Java (or Eiffel) with pre- and post-conditions and design a checker to verify that the pre and post conditions hold. What form the checker takes is an open question. A full-fledged certifying (or verifying) compiler is still a research activity, however, good progress was made [Ho03].

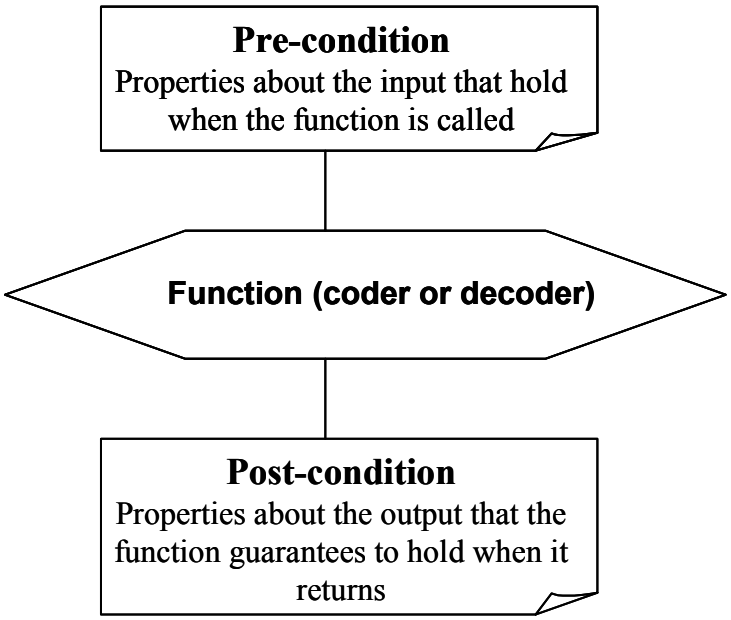

Figure 5.1 – Pre-and post-conditions for ASN.1 coder/decoder verification

6 Conclusion

ASN.1 is used in several important protocols deployed in commercial and non-commercial (for example, military) environments; therefore it is very important to ensure that ASN.1 implementations are safe. A number of vulnerabilities related to ASN.1 were found in implementations of SNMP and OpenSSL. Some vulnerabilities led to buffer overflows and caused the devices to fail. SNMP uses ASN.1 to describe MIB objects and OpenSSL for security certificates. It should be emphasized that the failures of SNMP and SSL implementations are caused by incorrect software designs that lack also robustness and not because of any flaw of the ASN.1 language.

Exploiting SNMP and OpenSSL vulnerabilities in an operating environment may lead to denials of services, since legitimate users will be denied access to the services provided until the network device recovers from the failure. Furthermore rogue entities may be tempted to mount active attacks to exploit any potential vulnerability.

Protocol implementations are part of a large and complex process known as protocol engineering; usually this process is not rigorously followed because of lack of expert resources, limited time and funding, even in large organizations.

Formal verification of protocol (and software) implementations is still difficult to conduct, especially with large-scale implementations. Checking the correctness of ASN.1 coders and decoders and ASN.1 compilers with assertions as pre- and post-conditions is a promising approach but it is still a challenge. The objective of this contribution is to create interest among researchers, developers, tool builders and vendors alike to design ASN.1 tools that ensure that ASN.1 implementations are safe to deploy.

7 References

[CERT1] CERT vulnerability note VU#854306, Multiple vulnerabilities in SNMPv1 request handling, http://www.kb.cert.org/vuls/id/854306.

[CERT2] CERT vulnerability note VU#107186, Multiple vulnerabilities in SNMPv1 trap handling, http://www.kb.cert.org/vuls/id/107186.

[CERT3] CERT vulnerability note VU#748355, ASN.1 parsing errors exist in implementations of SSL, TLS, S/MIME, PKCS#7 routines, http://www.kb.cert.org/vuls/id/748355.

[CLB00] Christopher Colby, Peter Lee, et. al., A certifying compiler for Java, ACM Sigplan Conference on programming language design and implementation 2000.

[Ho03] Tony Hoare, The verifying compiler: A grand challenge for computing research Journal of the ACM, Vol. 50, No. 1, January 2003, pp. 63-69.

[Jv] Programming with assertions, http://java.sun.com/j2se/1.4.1/docs/guide/lang/assert.html#usage.

[Ko99] Dexter Kozen, Language-based Security, in Proc. Mathematical Foundations of Computer Systems, 1999, vol. 1672 of LNCS, pp. 284-298, Springer-Verlag.

[McM00] Gary McGraw, Greg Morrissett, Attacking malicious code, IEEE software vol. 17, No. 5, September/October 2000, pp. 33-41.

[Me88] Bertrand Meyer, Object-oriented software construction, Prentice-Hall, 1988.

[Me92] Bertrand Meyer, Eiffel the language, Prentice-Hall, 1992.

[OSSL] http://www.openssl.org/.

[PROTOS] http://www.ee.oulu.fi/research/ouspg/protos/testing/c06/snmpv1/.

[RFC1067] IETF RFC 1067, A simple network management protocol, August 1988.

[RFC1155] IETF RFC 1155, Structure and identification of management information for TCP/IP-based internets, May 1990.

[RFC1156] IETF RFC 1156, Management Information base for network management of TCP/IP-based internets, May 1990.

[SMH] Fred B. Schneider, Greg Morrissett, Robert Harper, A Language-based approach to Security, Dagsthul 10th Anniversary volume, 2000.

[X509] ITU-T Recommendation X.500, Information technology – Open Systems Interconnection, The Directory: Public-key and attribute certificate frameworks.