Potential Pitfalls of the Use of ASN.1 in IETF Protocols

Status of This Memo

This document is an Internet-Draft and is in full conformance with all provisions of Section 10 of RFC2026.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

Comments on this document should be sent to the author.

Abstract

A number of IETF protocols make use of Abstract Syntax Notation One (ASN.1), which is a very complex language for describing abstract types and values, in addition to several sets of encoding rules for those types and values. Some of these uses of ASN.1, particularly the Kerberos protocol [RFC1510] pose implementation problems. This document analyzes some of the likely problems associated with the use of ASN.1 in IETF protocols, and some possible reasons for these problems.

Table of Contents

Status of This Memo ............................................ 1
Abstract .................................................... 1
Table of Contents ............................................ 1
1.  Introduction ............................................. 3
2.  Availability of Standards Documents ..................... 3
3.  Inefficiency of Encodings With Tagging ................... 3
   3.1.  A Brief Description of ASN.1 Tagged Types .......... 3
   3.2.  BER Encodings of Tagged Types ...................... 4
4.  Integer Encodings ....................................... 5
5.  Excessive Ease of Protocol Definition Changes .......... 6
6.  Case Studies ............................................ 6
6.1. Kerberos .................................................. 6
6.1.1. DER vs BER ........................................... 6
6.1.2. BitStringType Problems ............................. 7
6.1.3. GeneralString vs UTF8String ....................... 7
6.1.4. GeneralizedTime ..................................... 7
6.2. GSSAPI ..................................................... 8
7. References ................................................... 8
8. Author’s Address ............................................ 9
1. Introduction

The Abstract Syntax Notation One (ASN.1) [X.680, X.681, X.682, X.683, X.690, X.691] is a joint ITU-T and ISO/IEC standard defining a complex language for describing abstract types and values, as well as describing sets of rules for encoding these types and values. The use of this notation within the IETF is often a subject of debate, since many implementors encounter problems while attempting to handle ASN.1 syntax and encodings.

For now, this document assumes a reasonable degree of familiarity with the ASN.1 specifications, though this will change in future revisions where more background material regarding ASN.1 will be provided.

This is a very rough draft, and as such, many sections have not yet been fully written.

2. Availability of Standards Documents

One significant problem with the use of ASN.1 within an IETF context is that the standards documents that specify the notation and encoding rules are not available freely. Both the ITU-T and the ISO charge what many people believe to be exorbitant fees for either paper or electronic copies of the standards. It is not clear how many individuals who write IETF documents utilizing ASN.1 notation have actually read the official specifications describing the notation. This leads to problems when there are subtleties in the notation that are not obvious to someone who has only of third-party documents attempting to describe the notation and encodings.

3. Inefficiency of Encodings With Tagging

The Basic Encoding Rules (BER) and the Distinguished Encoding Rules (DER) are rather inefficient, though the actual abstract syntax describing the protocol can have significant impact on the verbosity of the encoding. Within the ASN.1 notation, there is a means of creating new types by tagging basic ASN.1 types.

3.1. A Brief Description of ASN.1 Tagged Types

A tagged type may be either explicitly or implicitly tagged. The type

FooInteger ::= [APPLICATION 0] EXPLICIT INTEGER

denotes an integer that is tagged explicitly with the "application 0" tag, while the type

BarInteger ::= [APPLICATION 1] IMPLICIT INTEGER

defines an integer that is tagged implicitly with the "application 1" tag. The most common usage of tags, however, is to distinguish
members of a complex type, e.g.

    FooPdu ::= SEQUENCE {
        version         [0] INTEGER,
        name            [1] UTF8String
    }

where the tags in this case belong to a class of tags known as "context-specific" tags.

If the type of tag (implicit vs explicit) is not specified, then the
TagDefault portion of ModuleDefinition gives one of three possibilities.

A TagDefault of EXPLICIT TAGS indicates that all tags are explicit.
This is the default if TagDefault is empty.

A TagDefault of IMPLICIT TAGS indicates that most tags are implicit.
The cases where the tags are not automatically made implicit are
somewhat complex and will not be discussed here.

A TagDefault of AUTOMATIC TAGS specifies a syntactic transformation
that automatically assigns implicit tags (in most cases) to all
members of composite types.

3.2. BER Encodings of Tagged Types

The encoding efficiency issue is that explicit tags, when encoded
with BER or DER, wrap an extra layer of encoding around the item in
question.  When types defined with ASN.1 are particularly complex and
contain many explicit tags, these extra layers of encoding add up
significantly.

A BER (or DER) encoding of a type consists of one or more identifier
octets followed by one or more length octets, and then the contents
octets.  The identifier octets describe the class of the tag
(universal, application, context-specific, private), its construction
(primitive or constructed), and its number.  The length octets merely
give the length of the enclosed contents octets.

Consider the following:

    FooInteger ::= [0] EXPLICIT INTEGER
    fooValue FooInteger ::= 1

This defines FooInteger as an explicitly tagged integer, with a
context-specific tag of 0.  The value fooValue is a FooInteger with a
value of 1.  This would be encoded as follows (in hexadecimal):

    A0 03 02 01 01

which can be broken down as follows:
If the value were instead:

```markdown
BarInteger ::= [0] IMPLICIT INTEGER
barValue ::= 1
```

then we would have the following encoding that would not include the tag for the IntegerType:

```markdown
80 01 01
```

which can be broken down as follows:

```markdown
identifier length contents
80 01 01
+---- context-specific 0 primitive
```

There is the disadvantage that using implicit tags with BER or DER result in encodings that a generic decoder cannot decode, but that is a tradeoff that needs to be evaluated by the protocol designer.

4. Integer Encodings

The encoding of an IntegerType is always signed in ASN.1, which means that there may be code that erroneously generates negative values when intending to encode values that have a leading one bit. Also, since an IntegerType must always be encoded in the fewest number of octets, it is possible to generate an erroneous encoding for even the integer "-1". The correct encoding of the integer

```markdown
fooInteger INTEGER ::= -1
```

is

```markdown
identifier length contents
02 01 FF
+---- universal 2 (INTEGER) primitive
```

rather than
which is not permitted because it is not minimal. Additionally, the integer

\[
\text{barInteger INTEGER ::= 255}
\]

must be encoded as

\[
\begin{array}{ccc}
\text{identifier} & \text{length} & \text{contents} \\
02 & 02 & 00 \text{ FF}
\end{array}
\]

in order to prevent it from being misread as the integer "-1".

5. Excessive Ease of Protocol Definition Changes

One theoretical benefit of using ASN.1 to describe protocols is that it is possible for the protocol designer to specify a protocol without regard to how it will be represented on the wire. Unfortunately, this feature is also a disadvantage, as it makes it difficult to tell when a change to the abstract syntax will result in a change in the wire protocol.

Many protocols are defined without making use of the ASN.1 notation allowing for extensibility of various composite types. This means that adding fields to composite types is a troublesome operation.

XXX need to add additional text about extensibility defaults, etc. and examples as well.

6. Case Studies

6.1. Kerberos

The Kerberos protocol [RFC1510] makes use of ASN.1 to define its protocol. There are a number of problems with the usage of ASN.1 in the protocol document, some of which are detailed here.

6.1.1. DER vs BER

One of the most significant problems in implementations of Kerberos is that while the specification dictates the use of DER only for the protocol, some implementations generate BER. The problem here is that fully general BER encodings are rather difficult to parse, particularly the ability to generate indefinite length encodings. This type of length encoding defers the specification of the length of an encoding to its contained encodings, which means that it is not possible to know from the beginning octets of an encoding where to expect the end of the encoding.

In one early MIT implementation of Kerberos, an ASN.1 compiler was used to generate the encoders and decoders. The code output by this
6.1.2. BitStringType Problems

In DER, the use of named bits within a declaration of a BitStringType mandates the removal of trailing zero bits from the encoding. In MIT Kerberos, the implementation always sent a 32-bit long bit string, which was incorrect. This subtle point is not necessarily obvious even after reading the ASN.1 specifications. The underlying reasons for this are not entirely clear, but an ASN.1 bit string has an inherent length that is part of its abstract value. A bit string that is declared without a named list of bits is purely a bit string of a definite number of bits, with a well-defined length. The problem with named bits in bit strings arises because it is inherently an overloading of the BitStringType notation to describe what amounts to a compact vector of boolean values. In order to construct a canonical encoding for this sort of bit string, it is necessary to remove trailing zero bits from the bit string.

Unfortunately, the DER constraints on BitStringType encodings require that even in the case where a size constraint indicates a bit string of a fixed size, the transmitted encoding must strip trailing zero bits anyway. This means that in order to permit interoperability with MIT Kerberos, it is necessary to avoid the use of the named bits notation within all usages of BitStringType notation in revisions of the Kerberos protocol.

6.1.3. GeneralString vs UTF8String

The Kerberos protocol specification uses GeneralString as its basic character string type. This may cause complications as it is essentially using ISO 2022 based escape sequences to invoke different character sets into the G0, G1, C0, and C1 parts of a seven or eight bit encoding. This is contrary to the general trend to using UTF-8 encodings in IETF protocols for internationalization. The methods of handling character set invocation by use of escape sequence are likely not widely understood or easily implementable at present, in addition to being potentially more restrictive than UTF-8, which can encode the entirety of the ISO 10646-1 character set.

6.1.4. GeneralizedTime

The Kerberos protocol makes use of the GeneralizedTime type in order to transmit date and time information. This has the disadvantage of being rather bulky, as it basically requires encoding the ASCII
A string corresponding to an ISO 8061 date format. This amounts to a total of 15 octets just to represent a date and time with a granularity of one second! Arguably an integer parameter that encodes the number of seconds since some epoch, as in the UNIX operating system, may be more compact, even if it is encoded as an ASN.1 integer. At least the fractional parts of seconds are indicated by an integer, rather than by adding further decimal digits to the GeneralizedTime in order to indicate fractional seconds.

6.2. GSSAPI

The Generic Security Service Application Program Interface (GSSAPI) [RFC2743] defines the generic initial token in a GSSAPI context establishment, the InitialContextToken, in terms of ASN.1. This definition actually uses tagging in an intelligent fashion, but describes the innerContext token field using a notation, the ANY type, which is deprecated in modern versions of the ASN.1 specification. Furthermore, the innerContextToken field in the initial token is not required to be encoded in ASN.1, which essentially prevents an unmodified ASN.1 parser from decoding such a token. The required use of DER at least makes the task of extracting the innerContextToken field possible at all, since the innerContextToken would likely be treated as trailing garbage by most ASN.1 decoders that haven’t been modified.

Requiring that the innerContextToken field of the InitialContextToken be encoded in a DER octet string in the case where the mechanism wishes to use a non-ASN.1 encoding of it data structures would make the InitialContextToken cabable of being parsed by an unmodified ASN.1 decoder.

7. References


8. Author’s Address

Tom Yu
Massachusetts Institute of Technology
Room E40-345
77 Massachusetts Avenue
Cambridge, MA 02139
USA

email: tlyu@mit.edu
phone: +1 617 253 1753