The 3rd IEEE International Symposium on Wireless Systems within the Conferences on Intelligent Data Acquisition and Advanced Computing Systems
26-27 September 2016, Offenburg, Germany

Transformation between XML and CBOR for network load reduction

Thomas Rix[1], Kai-Oliver Detken[1], Marcel Jahnke[1]
[1] DECOIT GmbH, Fahrenheitstr. 9, D-28359 Bremen, rix/detken/jahnke@decoit.de, www.decoit.de

Many systems use XML as a standardized information transfer between different components. The possibility to describe the data structure through definition documents enables both sender and receiver to validate information sent and received. However, the XML format requires relatively much bandwidth for a transfer via a network. This feature leads to a very high network load in case of applications with a huge amount of data to be transferred. The developed SIEM-like system from the SIMU project (www.simu-project.de) is based on the IF-MAP protocol and therefore uses the XML-based SOAP to represent and transfer data. To reduce network load, a technique was developed during the SIMU project, which allows a lossless transformation between XML and the relatively new specification called CBOR (RFC-7049). Thus, mobile terminals may be connected to such systems with high performance and low bandwidth usage.

Keywords: XML, IF-MAP, CBOR, trusted computing.

I. INTRODUCTION

Today the use of distributed systems is widespread. The *Extensible Markup Language (XML)* has become the de-facto standard for communication between individual components. The structure of XML documents can be defined and validated by various schema languages (e.g. DTD and XSD). Thus, components manufactured by different producers can communicate with one another without any problems as long as they all follow the previously determined specification of data structure. Moreover, for people it is possible to read these documents without converting or decoding them first. This means that XML documents do not necessarily have to be created mechanically and may be debugged without the use of specialized software.

However, these features bring along disadvantages. On one hand, XML elements and their attributes have usually clear names. They are easily readable but unnecessary long for the purpose of data transfer. The same applies to definition and description of namespaces, which are usually specified as complete URI and are referenced by tokens assigned to them. The latter is not required, a namespace may be defined individually for each single element. On the other hand, XML documents contain a large amount of control characters. And finally, it is a plain text format which requires more bytes then necessary to represent common data types, e.g. integers or IP addresses.

Therefore, XML is suitability for systems, which produce high amounts of single data entities in a short timeframe, is limited. Especially if the entities cannot be aggregated properly prior to network transfer the usage of XML adds a lot of overhead. SIEM systems can be hold up as an example, because their sensors send hundreds or even thousands of events per second to a central aggregator. In case of such large amounts of data, every saved byte helps to keep the network infrastructure in an operational state over the period of the data spike.

II. PROBLEM DESCRIPTION

In order to keep the advantages of XML in such systems, the bandwidth necessary for each transfer must be drastically reduced. Text-based formats require too much bandwidth because of their plain text representation and the trade-offs this implicates. Hence, a binary format is needed which allows a lossless conversion from and to XML documents.

The *Concise Binary Object Representation (CBOR, RFC-7049)* is a possible solution for this problem. As the name itself implies, this format was developed to keep the required overhead as small as possible. It is an extension of the JSON standard *(RFC-4627)* and as such it contains the *map* and *array* data types, which are necessary to represent structured data. By definition all data fields in CBOR are typed and thus data is stored in its original representation. For example an integer is stored in its actual byte representation and not as a sequence of digit characters.

The difficulty consists in transfer XML formatted data into a structure which conserves all information from XML the XML document but can be easily described through CBOR. Additionally, the XML document that is decoded from the CBOR representation must be semantically identical to the original document Apart from obvious information such as attributes, data types and values, additional structural information such as interleaving and sequence of data have to be properly conserved.

III. Concise Binary Object Representation (CBOR)

The *Concise Binary Object Representation (CBOR)* has been published by the Internet Engineering Task Force (IETF) in RFC-7049 [1]. Apart from the already mentioned *map* and *array* data types, the specification further defines six additional types which allow representation of any data item. Those data types are called “major types” and are numbered from 0 to 7 which corresponds their byte representation. Those eight types are:

a. Major Type 0: Unsigned Integer

b. Major Type 1: Negative Integer

c. Major Type 2: Byte-String

d. Major Type 3: UTF-8 Text-String

e. Major Type 4: Array

f. Major Type 5: Map

g. Major Type 6: Tag

h. Major Type 7: Floating Point and special simple data types

Major type 6 enables tagging of data items. Every major type, another tag as well, can be provided with a tag to make additional semantic information available for a decoder. An order of tags has been registered from IANA and has a fixed meaning for every CBOR decoder. And so, a string can be marked in a RFC-3339 date format by assigning the tag 0 to it. Furthermore, various data types such as bignum and bigfloat, codings as base64, and other data, which needs a special handling from the decoder, are marked. Additionally, tags can be defined according to specific needs.

A field in CBOR, consisting of type description and value, is called a *data item* and always has the same structure that requires at least one byte. The major type is encoded according to its number as the three most significant bits of the first byte of a data item. The remaining five bits describe additional information which interpretation depends on the major type.

Additional information values from 0 to 23 are interpreted as the stored value itself for *major types 0 and 1*. Values greater than 23 have special meanings and describe the length of the actual integer that is encoded in the following bytes. The additional information 24 stands for a length of 8 bit, 25 for 16 bit, 26 for 32 bit and 27 for 64 bit.

The *major types 2 and 3* for byte and Unicode strings use the same encoding of additional information as the major types 0 and 1. In this case the encoded integer inside the additional information and possibly following bytes describe the length of the string in bytes.

The data structures array and map, or *major type 4 and 5*, interpret their additional information the same way as the string types. The exact meaning is different for arrays and maps. For arrays it describes the number of elements (data items) inside the array and for maps the number of key-value pairs inside the map. A key-value pair is made up of two data items while the first is the key and the second is the value.

The *major type 6* represents a tag and codes the ID number of the tag in the same way as major type 0. Tags may be used to add additional semantic metadata to any data item, even other tags. The metadata can be used by the CBOR decoder to decide how the handle the contained information. For example a string tagged with the tag 0 should be interpreted as a RFC-3339 date format string by the decoder. Other tags mark various data types such as bignum and bigfloat and base64 encoded data. Pages 15 and 16 of [1] contain a table of standardized tags that registered with IANA. The table also states ID number ranges which are free for everyone to use to define their own tags.

The additional information for the *major type 7* defines what the actual data item represents. This is necessary since major type 7 may represent several different types of data. The values 0 to 23 mark simple data without content, e.g. *TRUE*, *FALSE*, *NULL*, and *UNDEFINED*. The values 25, 26, 27 describe for a floating point number which is encoded in the following bytes. The different values describe IEEE 754 half-, single- and double precision floats with the lengths of 16 bits, 32 bits or 64 bits respectively.

The string and data structure major types 2 to 5 can be marked as data items with unlimited length when they contain the additional information 31. For the exact structure of such infinite data items please refer to [1]. To close such a data type an additional data item of major type 7 with additional information 31 is used.

IV. TRANSLATION FROM XML INTO CBOR

In order to describe XML formatted data in CBOR, some problems, which result from different structure of XML and CBOR or JSON, have to be solved. Specifically they concern the following aspects of XML:

a. Preserve the sequence of elements

b. Namespaces

c. Attributes of elements

These three concepts are unknown to CBOR. Therefore, as a first step a structure has to be developed in which this information may be stored by not requiring too much space at the same time.

The following examples use a notation based on JSON to present data, because a native CBOR representation is hardly readable for humans. Numbers, which are followed by brackets enclosing a data item, describe CBOR tags.

The sequence of elements is important in XML. Because of this it has to be preserved in a CBOR data structure. The only available data type which can ensure that is the array. Thus, the outer structure has to be an array as it is shown in listing 1. This encapsulation of data guarantees that the primary sequence of elements in the XML document remains unchanged after decoding it from a CBOR data structure:

LISTING 1: TRANSLATION FROM XML INTO CBOR, STEP1

```
[
      <Element>,
      <Element>
]
```

The XML document described in listing 2 will serve as an example throughout this section:

LISTING 2: EXAMPLE OF XML ELEMENTS

```
<element attribute=”attr-value” xmlns=”some-namespace>
  <nested-element attribute=”attr-value” smlns=”some-
  namespace”/>
</element>
```

An XML element consists of four fundamental parts:

a. Namespace
b. Element name
c. Attributes with values
d. Value of elements or further elements

The easiest way to translate these features into CBOR is to consider attributes and their values as key-value pairs of a map and introduce special keys for element names and values (see listing 3):

LISTING 3: XML ELEMENT PRESENTATION WITH SPECIAL KEYS

```
{
  $name: "element"
  xmlns: "some-namespace",
  attribute: "attribute-value",
  $value: {
      $name: "nested-element",
      xmlns: "some-namespace",
      attribute: "attribute-value",
      $value: null
  }
}
```

However, this procedure has some drawbacks. On one hand, the special keys have to be chosen in such a way that they cannot collide inadvertently with actual attributes. This was guaranteed in the example through the usage of a leading dollar sign, which is not a valid character for an attribute name in XML.

The second disadvantage is that new keys appear in the generated data structure which are not essential and have never existed in the XML document. The only key-value pairs in XML are attributes and the namespace definition. So this is a step backwards, because the addition of new keys is opposed to the goal to ensure a memory imprint that is as small as possible.

A better and economical way is to present the features of XML elements as elements of an array. By convention of this procedure the first element in an array contains the namespace URI, the second one is the element name, the third one covers attributes and the fourth the element's value or further nested XML elements.

The attributes are represented as elements of an additional array. These form pairs whereas the first element of a pair is the name of the attribute and the second one is the value. Fundamentally, a map could be used instead of the attribute array. However, the usage of an array allows for a very slim CBOR generator and parser that does not even need to know the data type map.

As shown in listing 4, the resulting CBOR structure has no need for a namespace key which is a bit of information that can be saved in comparison to the XML document. Moreover, in contrast to the previously mentioned map solution, multiple elements can be concatenated without the need for a new surrounding array or map. Since a single XML element is always represented by exactly four array elements a single array may contain an unlimited number of XML elements.

LISTING 4: XML ELEMENT AS CBOR ARRAY STRUCTURE

```
[
  "some-namespace",
  "element",
  [
      "some-namespace",
      "nested-element",
      [
          "attribute",
          "attribute-value"
      ],
      null
  ]
]
```

To further reduce the required memory, values are stored in their natural representation instead of strings. Thus, XML attributes of the type xsd:date or xsd:datetime are converted into UTC time and added to the CBOR data structure as a UNIX timestamp. A timestamp is represented by an integer that was tagged with the CBOR tag 1 (epoch time). Storage as an integer instead of a string requires less bytes, especially if date and time have to be represented.

From the XML point of view decimal numbers are strings instead of floating point numbers. In order to transfer them into CBOR without a loss due to rounding errors, CBOR format decimal fraction is used (see page 17 in [1]). It consists of an array with two elements, the mantissa and the exponent with a base of 10. The array is marked with the CBOR tag 4. In this way it is ensured that no round-off errors appear during the conversion between XML and CBOR.

In order to represent IPv4, IPv6, and MAC addresses, additional CBOR tags have to be defined. Since they are not standardized, any random free tag number may be used. In the project SIMU the following ID numbers were applied: 40001 (IPv4), 40002 (IPv6), and 40003 (MAC).

These tags are used in conjunction with byte strings (major type 2) which contain the corresponding addresses in their respective byte representation. As with date and time formats, saving these addresses as byte strings can save a lot of memory in contrast to a string representation. This is important for SIEM systems, because most of the events generated by them contain one or more types of network addresses.

V. OPTIMIZATION THROUGH USAGE OF A DICTIONARY

The procedure of transforming XML documents into CBOR structures described in chapter 4 provides a possibility of saving bandwidth required in case of network transfers. However, a big part of optimization is equalized by the necessity of storing the namespace URI for every single element. XML allows definition of aliases in document header whereby there is no need to specify the whole namespace's URI for every element. The CBOR structure does not provide this option.

However, the optimization can be reproduced with aliases, which are defined in an additional *dictionary* (see figure 1). The dictionary allows translation between a short form and a complete namespace URI. But beyond that the

CBOR-XML dictionary allows many more possibilities to substitute fixed values in XML. Several other static parts of an XML document such as element names, attributes, and enum values may be replaced by aliases too. Since in this case CBOR is only a transport format, the only thing that has to be ensured is that the resulting XML is equivalent to the source XML document.

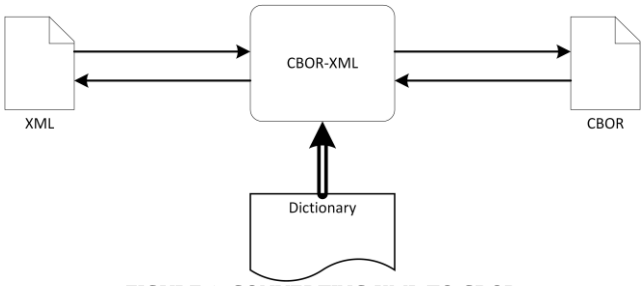

FIGURE 1: CONVERTING XML TO CBOR

As described in section III, CBOR allows to save integers from 0 to 23 as a single byte. Using this technique a huge reduction of memory usage can be achieved by translating static XML names into integers with the help of a dictionary.

In case of big documents, which contain more than 24 different static XML names, a second byte has to be used to describe the values of 24 and higher. This is not desirable. Therefore, the dictionary is arranged hierarchically. An alias has to be unambiguous on a respective hierarchy level only. Thereby, in most documents it is possible to handle the replacements with the 24 available one byte integers.

The dictionary is defined in a plain text format. Each line describes a static XML name and its CBOR alias. The data format is described by the EBNF shown in listing 5.

LISTING 5: ENBF OF DICTIONARY DEFINITION

```
DICTIONARY = NAMESPACE,
  {NAMESPACE} ;

NAMESPACE = "n'", NAMESPACEURI, "'[", CBORNAME, "]", [ "{",
    {TAG},
  "}" ] ;

TAG = "t'", XMLNAME, "'[", CBORNAME, "]", [ "{",
    {ATTRIBUTE},
    {TAG} | {ENUM},
  "}" ] ;

ATTRIBUTE = "a'", XMLNAME, "'[", CBORNAME, "]", [ "{",
    {ENUM},
  "}" ] ;

ENUM = "e'", XMLNAME, "'[", CBORNAME, "]" ;

NAMESPACEURI = ? valid XML namespace URI ?
XMLNAME = ? valid XML identifier (tag, attribute or enum name) ? ;
CBORNAME = UINT | NEGINT | DOUBLE | BYTESTR | UNISTR | BOOL ;

UINT = "uint(", ("0" | INTEGER), ")" ;
NEGINT = "negint(-", INTEGER, ")" ;
DOUBLE = "double(", FLOATINGPOINT, ")" ;
BYTESTR = "bytestr(", BYTESTRVAL, ")" ;
UNISTR = "unistr(", UTF8STRING, ")" ;
BOOL = "bool(", ("true" | "false"), ")" ;

INTEGER = ? positive integer excluding zero ? ;
FLOATINGPOINT = ["-"], INTEGER, ".", INTEGER ;
BYTESTRVAL = BYTELETTERPAIR, {BYTELETTERPAIR} ;
BYTELETTERPAIR = BYTELETTER, BYTELETTER ;
BYTELETTER = DIGIT | BYTECHAR ;
DIGIT = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9" ;
BYTECHAR = "A" | "B" | "C" | "D" | "E" | "F" ;
UTF8STRING = UTF8CHAR, {UTF8CHAR} ;
UTF8CHAR = ? any valid UTF-8 character excluding linebreaks ? ;
```

Static XML names may be substituted by one of the six CBOR data types: unsigned integer, negative integer, double, byte string, UTF-8 string, or Boolean. As mentioned above, the best compression can be achieved by using single byte data items such as unsigned or negative integer and Boolean. However, if desired the dictionary allows usage of more verbose substitutions to keep compressed CBOR structures more readable.

Listing 6 shows an exemplary dictionary definition for the already known XML document. Because it is stored as a plain text file, it can easily be exchanged between communicating parties. Every static XML name can be substituted by the integer 0 because they are all located on different hierarchy levels. Even different types of elements on the same hierarchy level, attributes and nested elements for example, can be replaced by the same integer because the CBOR structure stores them in different arrays.

LISTING 6: EXEMPLARY DICTIONARY

```
n'some-namespace'[uint(0)] {
 t'element'[uint(0)] {
  a'attribute'[uint(0)]
  t'nested-element'[uint(0)] {
   a'attribute'[uint(0)]
  }
 }
}
```

VI. APPLICATION EXAMPLE WITH IF-MAP

The SIEM-like system developed in the SIMU project applies the *Trusted Computing Group's* IF-MAP protocol ([4] and [5]) as a transport and storage format.

In *IF-MAP* entities of a monitored system are represented by so called identifiers. These can be servers, clients, infrastructure elements, or software services. Identifiers include only immutable and identifying information about an entity. Mutable attributes and relations between identifiers are described as metadata. In contrast to identifiers, metadata may be created, deleted or changed at any time. With these elements, identifiers and metadata, the current state of the monitored system can be as a graph in a MAP server (MAPS). Additionally the state can be queried from the MAPS. The sensors of the system are called MAP clients (MAPC) and keep the metadata up to date. Versioning of a MAP graph, which is not a feature that is included in the IF-MAP standard, allows to analyze changes in the system's state over time.

In order to transfer information between MAPC and a single MAPS, HTTP(S) connections are used. The transferred data is formatted as a *SOAP (Simple Object Access Protocol)* messaged and attached to the body of HTTP packets. Because SOAP is a special XML format, all previously mentioned disadvantages of XML regarding the data transfer apply to it.

In case of an actual network attack, the sensors of a SIEM system can create a huge amount of events in a very short timeframe. Usually all of these events have to be transferred to the analyzing component one by one. This puts a lot of

stress on the network and ultimately may lead to failures due to network overload.

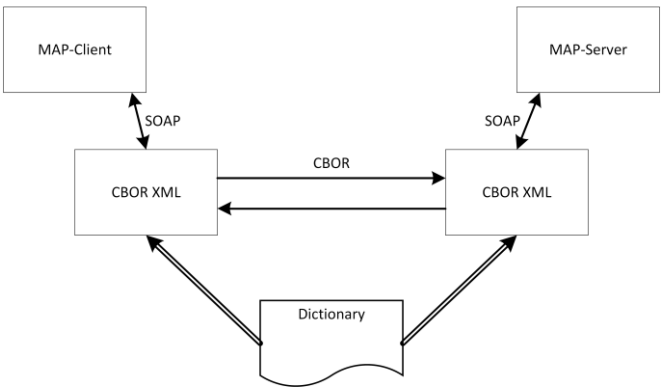

FIGURE 2: COMMUNICATION WITH CBOR

The procedure described in this paper was used in the project SIMU to avoid that problem and to reduce required bandwidth per data transfer. As MAP clients and MAP servers do not support CBOR natively, so called IF-MAP-CBOR proxies were used during the communication. They translate between SOAP and CBOR format and operate on a MAPC or MAPS machine locally. Thereby, the HTTP/SOAP protocol is required only for the local connection between a proxy and MAPC or MAPS. A socket connection is used between the proxies to reduce overhead that would otherwise be required to establish new HTTP connections for every data entity. Figure 2 shows how the CBOR-IF-MAP proxies were integrated into the IF-MAP communication.

VII. POSSIBLE SAVINGS IN IF-MAP DATA STRUCTURE

In order to present the effectiveness of the procedure, three exemplary IF-MAP structures from [4] were first generated as SOAP-XML and afterwards as a CBOR byte stream. The size in bytes of the resulting output then was compared. In the process the transformation in CBOR was conducted respectively with and without a dictionary to show the substitution effect on the amount of bytes. Furthermore, all data were compressed with GZIP to determine effectiveness of CBOR presentation in contrast to a GZIP compressed HTTP connection. Thereby, standard settings of GZIP were used. GZIP compression may be further increased by using the best possible settings but this has an impact on compression speed.

The test suite was developed in Java. The *ifmapj*[1] library from the Trust@HsH research group of the University of Applied Sciences Hanover has been used as IF-MAP implementation. To convert IF-MAP into CBOR[2] the Java libraries developed during the SIMU project were used. The same applies for the dictionary implementation[3]. Equally, the IF-MAP dictionary definition was applied, which was created as a part of the project and which describes all namespaces, XML elements, attributes, and enum values that exist in IF-MAP as single-byte unsigned integers. All libraries used for this test suite are available as open source software under the links given below. Using the respective mechanisms of the libraries the examples from the IF-MAP specification were implemented with data objects and then encoded by the library's emitter. Thus the resulting XML document already contains the required SOAP envelope elements and namespaces.

All tests were realized locally, why only the pure payload size was considered and thus any HTTP overhead was ignored. Moreover, ifmapj creates XML documents in which every element contains a namespace attribute. Theoretically, the XML documents could be minimized by using namespace aliases. Nevertheless, this implementation was used in the SIMU project and is one of the most complete libraries available for IF-MAP with Java, which makes it a good opponent for this comparision.

The chosen examples represent common operations which have to be conducted on a MAP server. The examples 1 and 2 deal primarily with tasks which are done by sensors of a SIEM system, namely adding and deleting metadata in MAP graphs. The third example resembles a search for metadata and identifiers in MAP graphs which is usually performed by the analyzing system components.

A. *Example 1: publish notify request*

As a first example, a *publish notify request* will be used. The XML document can be found in paragraph 3.9.2.2 of [4] on page 40. It contains an event metadata which should be attached to an IP address identifier.

The size of SOAP-XML, which is created thereby, amounts 712 bytes. By the use of GZIP it can be reduced of approx. 43%, which means 408 bytes.

Without the use of the dictionary, the CBOR byte string consists of 436 bytes which can be reduced of approx. 30% to 305 bytes while using GZIP. If the IF-MAP-CBOR dictionary is applied, the created byte string has only 87 bytes. This is equal to about 20% of the previous byte string and only 12.2% of the original XML document. However, a GZIP compression has a disadvantage just here and increases the byte string to a length of 107 bytes.

B. *Example 2: publish delete request*

The second example is a *publish delete request*. It can be found in paragraph 3.9.2.3 of [4] on page 41. It deletes all metadata which are attached to both the specified IP address and MAC address identifiers in a MAP graph.

The size of SOAP-XML amounts to 358 bytes and can be compressed by approx. 30% to 251 bytes.

The CBOR byte string consists of 256 bytes if the dictionary is not applied. It can be shortened to 167 bytes what corresponds with approx. 65%. If the dictionary is in use, the byte string can be reduced to only 44 bytes. This corresponds to approx. 17.2% of the CBOR byte string without using the dictionary and only 12.3% of the original

[1] https://github.com/trustathsh/ifmapj

[2] https://github.com/decoit/cbor-if-map-tnc-base

[3] https://github.com/decoit/cbor-xml-dictionary

XML document. As with the first example, a GZIP compression extended the byte string to a length 63 bytes.

C. Example 3: search request

An IF-MAP *search request* will serve as third example. It is required if an analyzing system component requests information from the MAP server. The XML document representing the request can be found in the paragraph 3.9.3.4 of [4] on page 45.

The SOAP-XML, which is generated by ifmapj, has a size of 506 bytes. Using GZIP it can be reduced to 331 bytes which is approx. 65.4% of its initial size.

The CBOR byte string has a length of 365 bytes if the dictionary is not in use. GZIP achieves a compression to 255 bytes which corresponds with a reduction of approx. 30%. Application of the dictionary shortens the byte string length to 168 bytes or approx. 46% of the original byte string and approx. 33.2% of the SOAP-XML. This time, the usage of GZIP achieves further compression to 141 bytes.

VIII. CONCLUSION

Table 1 contains a brief overview of the test results. One can observe that the transformation of SOAP-XMAL into CBOR is efficient, especially if a dictionary is used. Without a dictionary the compression achieved by the CBOR transformation is very similar to GZIP, which means that the latter would be the better solution because it requires less computational overhead.

The substitution of namespaces, element names, attributes, and enum values with unsigned integers allows a huge reduction of required bandwidth for most use cases. The main reason for this is the efficient feature of CBOR to store small integers as single byte entities. Because of the hierarchical arrangement of the dictionary it should be possible to use only the integers 0 to 23 for most XML documents. If required, the range of negative single byte integers (-1 to -24) may be used as well, which allows 24 further elements on each hierarchy level to be substituted by single byte entities.

TABLE I. PRESENTATION OF THE RESULTS

	SOAP-XML (gzip)	***CBOR (gzip)***	***CBOR with dictionary (gzip)***
Example 1	***712 bytes (408)***	***436 bytes (305)***	***87 bytes (107)***
Example 2	***358 bytes (251)***	***256 bytes (167)***	***44 bytes (63)***
Example 3	***506 bytes (331)***	***365 bytes (255)***	***168 bytes (141)***

The examples 1 and 2 represent typical use cases of a SIEM system sensor's network traffic, which was the main concern to develop this technique. Reducing the data to be transferred by approx. 88% per request is a huge improvement to solve the problem of network overload.

The, in contrast to example 1 and 2, relatively bad compression in example 3 is caused by the many free text attributes an IF-MAP search request contains. These texts cannot be substituted effectively and thus have to be encoded as UTF-8 strings in CBOR which results in comparatively long byte strings.

Moreover, the results indicate that an additional compression of communication channels by using GZIP can be counterproductive when a dictionary is in use. Apart from structures with much free text, such as the search request, an additional application of GZIP compression causes an increase of the CBOR byte string length. Thus, this is not advisable.

The results were discussed with the Trusted Computing Group (TCG) during and after the SIMU project's lifecycle in order to recommend CBOR as a new transfer protocol for IF-MAP. Currently some work has been done to embed CBOR into the new IF-MAP specification.

Acknowledgment

The authors give thanks the BMBF [6] for the financial support as well as all other partners involved in the research project SIMU for their great collaboration. The project consortium consisted of the industrial partners DECOIT GmbH, NCP engineering GmbH, macmon secure GmbH, and the research partners Fraunhofer SIT, and University of Applied Sciences and Arts of Hanover. A special appreciation goes to Fraunhofer SIT for their support during the development of the XML description in CBOR and making the CBOR-proxy available as well as to the University of Applied Sciences and Arts of Hanover for their extensive work and developments in the IF-MAP area.

References

[1] Carsten Bormann, Paul Hoffman: *Concise Binary Object Representation (CBOR)*. Internet Engineering Task Force, RFC-7049, ISSN: 2070-1721, October 2013

[2] D. Crockford: *The application/json Media Type for JavaScript Object Notation (JSON)*. Internet Engineering Task Force, RFC-4627, Network Working Group, July 2006

[3] G. Klyne, C. Newman: *Date and Time on the Internet: Timestamps*. Internet Engineering Task Force, RFC-3339, Network Working Group, July 2002

[4] Trusted Computing Group: *TNC IF-MAP Binding for SOAP Version 2.2 Revision 10*. Trusted Computing Group, Incorporated, März 2014

[5] Trusted Computing Group: *TNC IF-MAP Metadata for Network Security Version 1.1 Revision 9*. Trusted Computing Group, Incorporated, Mai 2012

[6] Federal Ministry of Education and Research: *http://www.bmbf.de/en/index.php*