                 Simple Network Management Protocol (SNMP)
                 Traffic Measurements and Trace Exchange Formats

Status of This Memo

IESG Note

   The IESG thinks that this work is related to IETF work done in the
   Operations and Management Area related to SNMP, but this does not
   prevent publishing. This RFC is not a candidate for any level of
   Internet Standard. The IETF disclaims any knowledge of the fitness
   of this RFC for any purpose and notes that the decision to publish is
   not based on IETF review apart from the IETF Last Call on the
   allocation of a URI by IANA and the IESG review for conflict with
   IETF work. The RFC Editor has chosen to publish this document at its
   discretion. See RFC 3932 for more information.

Abstract

   The Simple Network Management Protocol (SNMP) is widely deployed to
   monitor, control, and (sometimes also) configure network elements.
   Even though the SNMP technology is well documented, it remains
   relatively unclear how SNMP is used in practice and what typical SNMP
   usage patterns are.

   This document describes an approach to carrying out large-scale SNMP
   traffic measurements in order to develop a better understanding of
   how SNMP is used in real-world production networks. It describes the
   motivation, the measurement approach, and the tools and data formats
   needed to carry out such a study.

   This document was produced within the IRTF's Network Management
   Research Group (NMRG), and it represents the consensus of all of the
   active contributors to this group.

Table of Contents

1. Introduction

   The Simple Network Management Protocol (SNMP) was introduced in the
   late 1980s [RFC1052] and has since then evolved to what is known
   today as the SNMP version 3 Framework (SNMPv3) [RFC3410]. While SNMP
   is widely deployed, it is not clear what protocol versions are being
   used, which protocol features are being used, how SNMP usage differs
   in different types of networks or organizations, which information is
   frequently queried, and what typical SNMP interaction patterns occur
   in real-world production networks.

   There have been several publications in the recent past dealing with
   the performance of SNMP in general [SM99][Mal02][Pat01], the impact
   of SNMPv3 security [DSR01][CT04], or the relative performance of SNMP
   compared to Web Services [PDMQ04][PFGL04]. While these papers are
   generally useful to better understand the impact of various design
   decisions and technologies, some of these papers lack a strong
   foundation because authors typically assume certain SNMP interaction
   patterns without having experimental evidence that the assumptions
   are correct. In fact, there are many speculations on how SNMP is
   being used in real-world production networks, and performance
   comparisons are based on limited test cases, but no systematic
   measurements have been performed and published so far.

   Many authors use the ifTable of the IF-MIB [RFC2863] or the
   tcpConnTable of the TCP-MIB [RFC4022] as a starting point for their
   analysis and comparison. Despite the fact that there is no evidence
   that operations on these tables dominate SNMP traffic, it is even
   more unclear how these tables are read and which optimizations are
   done (or not done) by real-world applications. It is also unclear
   what the actual traffic trade-off between periodic polling and more
   aperiodic bulk data retrieval is. Furthermore, we do not generally
   understand how much traffic is devoted to standardized MIB objects
   and how much traffic deals with proprietary MIB objects and whether
   the operation mix between these object classes differs between
   different operational environments (e.g., backbone networks, access
   networks, enterprise networks).

   This document recommends an approach to collecting, codifying, and
   handling SNMP traffic traces in order to find answers to some of
   these questions. It describes the tools that have been developed to
   allow network operators to collect traffic traces and to share them
   with research groups interested in analyzing and modeling network
   management interactions.

   While the SNMP trace collection and analysis effort was initiated by
   the research community, network operators can benefit from the SNMP
   measurements too. Several new tools are being developed as part of

this effort that can be used to capture and analyze the traffic
generated by management stations. This resulting information can
then be used to improve the efficiency and scalability of management
systems.

The measurement approach described in this document is by design
limited to the study of SNMP traffic. Studies of other management
protocols or the impact of management protocols such as SNMP on other
traffic sharing the same network resources is left to future efforts.

This is an Informational document, produced within the IRTF's Network
Management Research Group (NMRG), and it represents the consensus of
all of the active contributors to this group.

2. Measurement Approach

   This section outlines the process of doing SNMP traffic measurements
   and analysis. The process consists of the following five basic
   steps:

   1. Capture raw SNMP traffic traces in pcap packet capture files [1].

   2. Convert the raw traffic traces into a structured machine and
       human-readable format. A suitable XML schema has been developed
       for this purpose that captures all SNMP message details. Another
       more compact comma-separated values (CSV) format has been
       developed that only keeps key information about SNMP messages.

   3. Filter the converted traffic traces to hide or anonymize
       sensitive information. While the filtering is conceptually a
       separate step, filtering may actually be implemented as part of
       the previous data conversion step for efficiency reasons.

   4. Submit the filtered traffic traces to a repository from which
       they can be retrieved and analyzed. Such a repository may be
       public, under the control of a research group, or under the
       control of a network operator who commits to run analysis scripts
       on the repository on behalf of researchers.

   5. Analyze the traces by creating and executing analysis scripts
       that extract and aggregate information.

   Several of the steps listed above require the involvement of network
   operators supporting the SNMP measurement projects. In many cases,
   the filtered XML and CSV representation of the SNMP traces will be
   the interface between the researchers writing analysis scripts and
   the operators involved in the measurement activity. It is therefore
   important to have a well-defined specification of these interfaces.

This section provides some advice and concrete hints on how the steps
listed above can be carried out efficiently. Some special tools have
been developed to assist network operators and researchers so that
the time spent on supporting SNMP traffic measurement projects is
limited. The following sections describe the five steps and some
tools in more detail.

2.1. Capturing Traffic Traces

Capturing SNMP traffic traces can be done using packet sniffers such
as tcpdump [2], wireshark [3], or similar applications. Some care
must be taken to specify pcap filter expressions that match the SNMP
transport endpoints used to carry SNMP traffic (typically 'udp and
(port 161 or port 162)'). Furthermore, it is necessary to ensure
that full packets are captured, that is packets are not truncated
(tcpdump option -s 0). Finally, it is necessary to carefully select
the placement of the capturing probe within the network. Especially
on bridged LANs, it is important to ensure that all management
traffic is captured and that the probe has access to all virtual LANs
carrying management traffic. This usually requires placing the
probe(s) close to the management system(s) and configuring dedicated
monitoring ports on bridged networks. Some bridges have restrictions
concerning their monitoring capabilities, and this should be
investigated and documented where necessary.

It is recommended to capture at least a full week of data to capture
diurnal patterns and one cycle of weekly behavior. Operators are
strongly encouraged to capture traces over even longer periods of
time. Tools such as tcpdump and tcpslice [2] or mergecap and
editcap [3] can be used to split or merge pcap trace files as needed.

Several operating systems can offload some of the TCP/IP processing
such as the calculation of transport layer checksum to network
interface cards. Traces that include traffic to/from a capturing
interface that supports TCP/IP offloading can include incorrect
transport layer checksums. The simplest solution is of course to
turn checksum offloading off while capturing traces (if that is
feasible without losing too many packets). The other solution is to
correct or ignore checksums during the subsequent conversion of the
raw pcap files.

It is important to note that the raw pcap files should ideally be
kept in permanent storage (e.g., compressed and encrypted on a CD ROM
or DVD). To verify measurements, it might be necessary to go back to
the original pcap files if, for example, bugs in the tools described
below have been detected and fixed.

For each captured trace, some meta data should be recorded and made
available. The meta data should include information such as where
the trace was collected (name of the network and name of the
organization owning the network, description of the measurement point
in the network topology where the trace was collected), when it was
collected, contact information, the size of the trace, any known
special events, equipment failures, or major infrastructure changes
during the data collection period and so on. It is also extremely
useful to provide a unique identification. There are special online
services such as DatCat [4] where meta data can be stored and which
provide unique identifiers.

2.2. Converting Traffic Traces

Raw traces in pcap format must be converted into a format that is
human readable while also remaining machine readable for efficient
post-processing. Human readability makes it easy for an operator to
verify that no sensitive data is left in a trace while machine
readability is needed to efficiently extract relevant information.

The natural choice here is to use an XML format since XML is human as
well as machine readable and there are many tools and high-level
scripting language application programming interfaces (APIs) that can
be used to process XML documents and to extract meaningful
information. However, XML is also pretty verbose, which increases
processing overhead. In particular, the usage of XML streaming APIs
is strongly suggested since APIs that require an in-memory
representation of XML documents do not handle large traces well.

Section 4.1 of this document defines a RELAX NG schema [OASISRNG] for
representing SNMP traffic traces in XML. The schema captures all
relevant details of an SNMP message in the XML format. Note that the
XML format retains some information about the original ASN.1/BER
encoding to support message size analysis.

A lightweight alternative to the full-blown XML representation based
on comma-separated values (CSV) is defined in Section 4.2. The CSV
format only captures selected parts of SNMP messages and is thus more
compact and faster to process.

As explained in the previous sections, analysis programs that process
raw pcap files should have an option to ignore incorrect checksums
caused by TCP/IP offloading. In addition, analysis programs that
process raw pcap files should be able to perform IP reassembly for
SNMP messages that were fragmented at the IP layer.

The snmpdump [5] package has been developed to convert raw pcap files
into XML and CSV format. The snmpdump program reads pcap, XML, or
CSV files as input and produces XML files or CSV files as output.

Specific elements can be filtered as required to protect sensitive
data.

2.3. Filtering Traffic Traces

Filtering sensitive data (e.g., access control lists or community
strings) can be achieved by manipulating the XML representation of an
SNMP trace. Standard XSLT processors (e.g., xsltproc [6]) can be
used for this purpose. People familiar with the scripting language
Perl might be interested in choosing a suitable Perl module to
manipulate XML documents [7].

The snmpdump program, for example, can filter out sensitive
information, e.g., by deleting or clearing all XML elements whose
name matches a regular expression. Data type specific anonymization
transformations that maintain lexicographic ordering for values that
appear in instance identifiers [HS06] can be applied. Note that
anonymization transformations are often bound to an initialization
key and depend on the data being anonymized in an anonymization run.
As a consequence, users must be careful when they merge data from
independently anonymized traces. More information about network
traffic trace anonymization techniques can be found in [XFA02],
[FXAM04], [PAPL06], and [RW07].

2.4. Storing Traffic Traces

The raw pcap traces as well as the XML / CSV formatted traces should
be stored in a stable archive or repository. Such an archive or
repository might be maintained by research groups (e.g., the NMRG),
network operators, or both. It is of key importance that captured
traces are not lost or modified as they may form the basis of future
research projects and may also be needed to verify published research
results. Access to the archive might be restricted to those who have
signed some sort of a non-disclosure agreement.

While this document recommends that raw traces should be kept, it
must be noted that there are situations where this may not be
feasible. The recommendation to keep raw traces may be ignored, for
example, to comply with data-protection laws or to protect a network
operator from being forced to provide the data to other
organizations.

Lossless compression algorithms embodied in programs such as gzip or
bzip2 can be used to compress even large trace files down to a size
where they can be burned on DVDs for cheap long-term storage.

It must be stressed again that it is important to keep the original
pcap traces in addition to the XML/CSV formatted traces since the
pcap traces are the most authentic source of information.
Improvements in the tool chain may require going back to the original
pcap traces and rebuilding all intermediate formats from them.

2.5. Analyzing Traffic Traces

Scripts that analyze traffic traces must be verified for correctness.
Ideally, all scripts used to analyze traffic traces will be
publically accessible so that third parties can verify them.
Furthermore, sharing scripts will enable other parties to repeat an
analysis on other traffic traces and to extend such analysis scripts.
It might be useful to establish a common, versioning repository for
analysis scripts.

Due to the availability of XML parsers and the simplicity of the CSV
format, trace files can be processed with tools written in almost any
programming language. However, in order to facilitate a common
vocabulary and to allow operators to easily read scripts they execute
on trace files, it is suggested that analysis scripts be written in
scripting languages such as Perl using suitable Perl modules to
manipulate XML documents <http://perl-xml.sourceforge.net/faq/>.
Using a scripting language such as Perl instead of system programming
languages such as C or C++ has the advantage of reducing development
time and making scripts more accessible to operators who may want to
verify scripts before running them on trace files that may contain
sensitive data.

The snmpdump tool provides an API to process SNMP messages in C/C++.
While the coding of trace analysis programs in C/C++ should in
general be avoided for code readability, verifiability, and
portability reasons, using C/C++ might be the only option in dealing
with very large traces efficiently.

Any results produced by analyzing a trace must be interpreted in the
context of the trace. The nature of the network, the attachment
point used to collect the trace, the nature of the applications
generating SNMP traffic, or the events that happened while the trace
was collected clearly influence the result. It is therefore
important to be careful when drawing general conclusions based on a
potentially (too) limited data set.

3. Analysis of Traffic Traces

   This section discusses several questions that can be answered by
   analyzing SNMP traffic traces. The questions raised in the following
   subsections are meant to be illustrative and no attempt has been made
   to provide a complete list.

3.1. Basic Statistics

   Basic statistics cover things such as:

   o protocol version used,

   o protocol operations used,

   o message size distribution,

   o error message type frequency, or

   o usage of authentication and encryption mechanisms.

   The Object Identifier (OID) names of the objects manipulated can be
   categorized into OID subtrees, for example, to identify
   'standardized', 'proprietary', and 'experimental' objects.

3.2. Periodic versus Aperiodic Traffic

   SNMP is used to periodically poll devices as well as to retrieve
   information at the request of an operator or application. The
   periodic polling leads to periodic traffic patterns while on-demand
   information retrieval causes more aperiodic traffic patterns. It is
   worthwhile to understand what the relationship is between the amount
   of periodic and aperiodic traffic. It will be interesting to
   understand whether there are multiple levels of periodicity at
   different time scales.

   Periodic polling behavior may be dependent on the application and
   polling engine it uses. For example, some management platforms allow
   applications to specify how long polled values may be kept in a cache
   before they are polled again. Such optimizations need to be
   considered when analyzing traces for periodic and aperiodic traffic.

3.3. Message Size and Latency Distributions

   SNMP messages are size constrained by the transport mappings used and
   the buffers provided by the SNMP engines. For the further evolution
   of the SNMP framework, it would be useful to know what the actual
   message size distributions are. It would be useful to understand the

latency distributions, especially the distribution of the processing
times by SNMP command responders. Some SNMP implementations
approximate networking delays by measuring request-response times,
and it would be useful to understand to what extent this is a viable
approach.

Some SNMP implementations update their counters from the underlying
instrumentation following adaptive algorithms, not necessarily
periodically, and not necessarily on-demand. The granularity of
internal counter updates may impact latency measurements and should
be taken into account.

3.4. Concurrency Levels

SNMP allows management stations to retrieve information from multiple
agents concurrently. It will be interesting to identify what the
typical concurrency level is that can be observed on production
networks or whether management applications prefer more sequential
ways of retrieving data.

Furthermore, it will be interesting to analyze how many redundant
requests coming from applications are processed almost simultaneously
by a device. The concurrency level and the amount of redundant
requests has implications on caching strategies employed by SNMP
agents.

3.5. Table Retrieval Approaches

Tables can be read in several different ways. The simplest and most
inefficient approach is to retrieve tables object-by-object in
column-by-column order. More advanced approaches try to read tables
row-by-row or even multiple-rows-by-multiple-rows. The retrieval of
index elements can be suppressed in most cases or only a subset of
columns of a table are retrieved. It will be useful to know which of
these approaches are used on production networks since this has a
direct implication on agent implementation techniques and caching
strategies.

3.6. Trap-Directed Polling - Myths or Reality?

SNMP is built around a concept called trap-directed polling.
Management applications are responsible to periodically poll SNMP
agents to determine their status. In addition, SNMP agents can send
traps to notify SNMP managers about events so that SNMP managers can
adapt their polling strategy and basically react faster than normal
polling would allow.

Analysis of SNMP traffic traces can identify whether trap-directed
polling is actually deployed. In particular, the question that
should be addressed is whether SNMP notifications lead to changes in
the short-term polling behavior of management stations. In
particular, it should be investigated to what extent SNMP managers
use automated procedures to track down the meaning of the event
conveyed by an SNMP notification.

3.7. Popular MIB Definitions

An analysis of object identifier prefixes can identify the most
popular MIB modules and the most important object types or
notification types defined by these modules. Such information would
be very valuable for the further maintenance and development of these
and related MIB modules.

3.8. Usage of Obsolete Objects

Several objects from the early days have been obsoleted because they
cannot properly represent today's networks. A typical example is the
ipRouteTable that was obsoleted because it was not able to represent
classless routing, introduced and deployed on the Internet in 1993.
Some of these obsolete objects are still mentioned in popular
publications as well as research papers. It will be interesting to
find out whether they are also still used by management applications
or whether management applications have been updated to use the
replacement objects.

Depending on the data recorded in a trace, it might be possible to
determine the age of devices by looking at the values of objects such
as sysObjectID and sysDecr [RFC3418]. The age of a device can then
be taken into consideration when analyzing the use of obsolete and
deprecated objects.

3.9. Encoding Length Distributions

It will be useful to understand the encoding length distributions for
various data types. Assumptions about encoding length distributions
are sometimes used to estimate SNMP message sizes in order to meet
transport and buffer size constraints.

3.10. Counters and Discontinuities

Counters can experience discontinuities [RFC2578]. A widely used
discontinuity indicator is the sysUpTime scalar of the SNMPv2-MIB
[RFC3418], which can be reset through a warm start to indicate
counter discontinuities. Some MIB modules introduce more specific
discontinuity indicators, e.g., the ifCounterDiscontinuityTime of the

IF-MIB [RFC2863]. It will be interesting to study to what extent
these objects are actually used by management applications to handle
discontinuity events.

3.11. Spin Locks

Cooperating command generators can use advisory locks to coordinate
their usage of SNMP write operations. The snmpSetSerialNo scalar of
the SNMPv2-MIB [RFC3418] is the default coarse-grain coordination
object. It will be interesting to find out whether there are command
generators that coordinate themselves using these spin locks.

3.12. Row Creation

Row creation is an operation not natively supported by the protocol
operations. Instead, conceptual tables supporting row creation
typically provide a control column that uses the RowStatus textual
convention defined in the SNMPv2-TC [RFC2579] module. The RowStatus
itself supports different row creation modes, namely createAndWait
(dribble-mode) and createAndGo (one-shot mode). Different approaches
can be used to derive the instance identifier if it does not have
special semantics associated with it. It will be useful to study
which of the various row creation approaches are actually used by
management applications on production networks.

4. Trace Exchange Formats

4.1. XML Representation

The XML format has been designed to keep all information associated
with SNMP messages. The format is specified in RELAX NG compact
notation [OASISRNC]. Freely available tools such as trang [8] can be
used to convert RELAX NG compact syntax to other XML schema
notations.

The XML format can represent SNMPv1, SNMPv2c, and SNMPv3 messages.
In case a new version of SNMP is introduced in the future or existing
SNMP versions are extended in ways that require changes to the XML
format, a new XML format with a different namespace needs to be
defined (e.g., by incrementing the version number included in the
namespace URI).

```
# Relax NG grammar for the XML SNMP trace format.
#
# Published as part of RFC 5345.
```

```
default namespace = "urn:ietf:params:xml:ns:snmp-trace-1.0"

start =
  element snmptrace {
    packet.elem*
  }

packet.elem =
  element packet {
    element time-sec  { xsd:unsignedInt },
    element time-usec { xsd:unsignedInt },
    element src-ip    { ipaddress.type },
    element src-port  { xsd:unsignedInt },
    element dst-ip    { ipaddress.type },
    element dst-port  { xsd:unsignedInt },
    snmp.elem
  }

snmp.elem =
  element snmp {
    length.attrs?,
    message.elem
  }

message.elem =
  element version   { length.attrs, xsd:int },
  element community { length.attrs, xsd:hexBinary },
  pdu.elem

message.elem |=
  element version { length.attrs, xsd:int },
  element message {
    length.attrs,
    element msg-id          { length.attrs, xsd:unsignedInt },
    element max-size        { length.attrs, xsd:unsignedInt },
    element flags           { length.attrs, xsd:hexBinary },
    element security-model  { length.attrs, xsd:unsignedInt }
  },
  usm.elem?,
  element scoped-pdu {
    length.attrs,
    element context-engine-id { length.attrs, xsd:hexBinary },
    element context-name      { length.attrs, xsd:string },
    pdu.elem
  }

usm.elem =
  element usm {
```

```
      length.attrs,
      element auth-engine-id    { length.attrs, xsd:hexBinary },
      element auth-engine-boots { length.attrs, xsd:unsignedInt },
      element auth-engine-time  { length.attrs, xsd:unsignedInt },
      element user              { length.attrs, xsd:hexBinary },
      element auth-params       { length.attrs, xsd:hexBinary },
      element priv-params       { length.attrs, xsd:hexBinary }
   }

pdu.elem =
   element trap {
      length.attrs,
      element enterprise        { length.attrs, oid.type },
      element agent-addr        { length.attrs, ipv4address.type },
      element generic-trap      { length.attrs, xsd:int },
      element specific-trap     { length.attrs, xsd:int },
      element time-stamp        { length.attrs, xsd:int },
      element variable-bindings { length.attrs, varbind.elem* }
   }

pdu.elem |=
   element (get-request | get-next-request | get-bulk-request |
            set-request | inform-request | snmpV2-trap |
            response | report) {
      length.attrs,
      element request-id        { length.attrs, xsd:int },
      element error-status      { length.attrs, xsd:int },
      element error-index       { length.attrs, xsd:int },
      element variable-bindings { length.attrs, varbind.elem* }
   }

varbind.elem =
   element varbind { length.attrs, name.elem, value.elem }

name.elem =
   element name { length.attrs, oid.type }

value.elem =
   element null              { length.attrs, empty } |
   element integer32         { length.attrs, xsd:int } |
   element unsigned32        { length.attrs, xsd:unsignedInt } |
   element counter32         { length.attrs, xsd:unsignedInt } |
   element counter64         { length.attrs, xsd:unsignedLong } |
   element timeticks         { length.attrs, xsd:unsignedInt } |
   element ipaddress         { length.attrs, ipv4address.type } |
   element octet-string      { length.attrs, xsd:hexBinary } |
   element object-identifier { length.attrs, oid.type } |
   element opaque            { length.attrs, xsd:hexBinary } |
```

```
  element no-such-object    { length.attrs, empty } |
  element no-such-instance  { length.attrs, empty } |
  element end-of-mib-view   { length.attrs, empty }

# The blen attribute indicates the number of octets used by the BER
# encoded tag / length / value triple.  The vlen attribute indicates
# the number of octets used by the BER encoded value alone.

length.attrs =
  ( attribute blen { xsd:unsignedShort },
    attribute vlen { xsd:unsignedShort } )?

oid.type =
  xsd:string {
    pattern =
      "(((0-1](\.[1-3]?[0-9]))|(2.(0|([1-9]\d*))))" ~
      "(\.(0|([1-9]\d*))){0,126}"
  }

# The types below are for IP addresses.  Note that SNMP's buildin
# IpAddress type only supports IPv4 addresses; IPv6 addresses are only
# introduced to cover SNMP over IPv6 endpoints.

ipv4address.type =
  xsd:string {
    pattern =
      "((0|(1[0-9]{0,2})" ~
      "|(2(([0-4][0-9]?)|(5[0-5]?)|([6-9]?)))|([3-9][0-9]?))\.){3}" ~
      "(0|(1[0-9]{0,2})" ~
      "|(2(([0-4][0-9]?)|(5[0-5]?)|([6-9]?)))|([3-9][0-9]?))"
  }

ipv6address.type =
  xsd:string {
    pattern =
      "(([0-9a-fA-F]+:){7}[0-9a-fA-F]+)|" ~
      "(([0-9a-fA-F]+:)*[0-9a-fA-F]+)?::(([0-9a-fA-F]+:)*[0-9a-fA-F]+)?"
  }

ipaddress.type = ipv4address.type | ipv6address.type
```

   The following example shows an SNMP trace file in XML format
   containing an SNMPv1 get-next-request message for the OID
   1.3.6.1.2.1.1.3 (sysUpTime) and the response message returned by the
   agent.

```
   <snmptrace xmlns="urn:ietf:params:xml:ns:snmp-trace-1.0">
     <packet>
       <time-sec>1147212206</time-sec>
       <time-usec>739609</time-usec>
       <src-ip>192.0.2.1</src-ip>
       <src-port>60371</src-port>
       <dst-ip>192.0.2.2</dst-ip>
       <dst-port>12345</dst-port>
       <snmp blen="42" vlen="40">
         <version blen="3" vlen="1">1</version>
         <community blen="8" vlen="6">7075626c6963</community>
         <get-next-request blen="29" vlen="27">
           <request-id blen="6" vlen="4">1804289383</request-id>
           <error-status blen="3" vlen="1">0</error-status>
           <error-index blen="3" vlen="1">0</error-index>
           <variable-bindings blen="15" vlen="13">
             <varbind blen="13" vlen="11">
               <name blen="9" vlen="7">1.3.6.1.2.1.1.3</name>
               <null blen="2" vlen="0"/>
             </varbind>
           </variable-bindings>
         </get-next-request>
       </snmp>
     </packet>
     <packet>
       <time-sec>1147212206</time-sec>
       <time-usec>762891</time-usec>
       <src-ip>192.0.2.2</src-ip>
       <src-port>12345</src-port>
       <dst-ip>192.0.2.1</dst-ip>
       <dst-port>60371</dst-port>
       <snmp blen="47" vlen="45">
         <version blen="3" vlen="1">1</version>
         <community blen="8" vlen="6">7075626c6963</community>
         <response blen="34" vlen="32">
           <request-id blen="6" vlen="4">1804289383</request-id>
           <error-status blen="3" vlen="1">0</error-status>
           <error-index blen="3" vlen="1">0</error-index>
           <variable-bindings blen="20" vlen="18">
             <varbind blen="18" vlen="16">
               <name blen="10" vlen="8">1.3.6.1.2.1.1.3.0</name>
               <unsigned32 blen="6" vlen="4">26842224</unsigned32>
             </varbind>
           </variable-bindings>
         </response>
       </snmp>
     </packet>
   </snmptrace>
```

4.2. CSV Representation

   The comma-separated values (CSV) format has been designed to capture
   only the most relevant information about an SNMP message. In
   situations where all information about an SNMP message must be
   captured, the XML format defined above must be used. The CSV format
   uses the following fields:

   1.   Timestamp in the format seconds.microseconds since 1970, for
         example, "1137764769.425484".

   2.   Source IP address in dotted quad notation (IPv4), for example,
         "192.0.2.1", or compact hexadecimal notation (IPv6), for
         example, "2001:DB8::1".

   3.   Source port number represented as a decimal number, for example,
         "4242".

   4.   Destination IP address in dotted quad notation (IPv4), for
         example, "192.0.2.1", or compact hexadecimal notation (IPv6),
         for example, "2001:DB8::1".

   5.   Destination port number represented as a decimal number, for
         example, "161".

   6.   Size of the SNMP message (a decimal number) counted in octets,
         for example, "123". The size excludes all transport, network,
         and link-layer headers.

   7.   SNMP message version represented as a decimal number. The
         version 0 stands for SNMPv1, 1 for SNMPv2c, and 3 for SNMPv3,
         for example, "3".

   8.   SNMP protocol operation indicated by one of the keywords get-
         request, get-next-request, get-bulk-request, set-request, trap,
         snmpV2-trap, inform-request, response, report.

   9.   SNMP request-id in decimal notation, for example, "1511411010".

   10.   SNMP error-status in decimal notation, for example, "0".

   11.   SNMP error-index in decimal notation, for example, "0".

   12.   Number of variable-bindings contained in the varbind-list in
         decimal notation, for example, "5".

   13.   For each varbind in the varbind list, three output elements are
         generated:

1. Object name given as object identifier in dotted decimal
    notation, for example, "1.3.6.1.2.1.1.3.0".

2. Object base type name or exception name, that is one of the
    following: null, integer32, unsigned32, counter32,
    counter64, timeticks, ipaddress, octet-string, object-
    identifier, opaque, no-such-object, no-such-instance, and
    end-of-mib-view.

3. Object value is printed as a number if the underlying base
    type is numeric. An IPv4 addresses is rendered in the
    dotted quad notation and an IPv6 address is rendered in the
    usual hexadecimal notation. An octet string value is
    printed in hexadecimal format while an object identifier
    value is printed in dotted decimal notation. In case of an
    exception, the object value is empty.

Note that the format does not preserve the information needed to
understand SNMPv1 traps. It is therefore recommended that
implementations be able to convert the SNMPv1 trap format into the
trap format used by SNMPv2c and SNMPv3, according to the rules
defined in [RFC3584]. The activation of trap format conversion
should be the user's choice.

The following example shows an SNMP trace file in CSV format
containing an SNMPv1 get-next-request message for the OID
1.3.6.1.2.1.1.3 (sysUpTime) and the response message returned by the
agent. (Note that the example uses backslash line continuation marks
in order to fit the example into the RFC format. Backslash line
continuations are not part of the CSV format.)

```
1147212206.739609,192.0.2.1,60371,192.0.2.2,12345,42,1,\
  get-next-request,1804289383,0,0,1,1.3.6.1.2.1.1.3,null,
1147212206.762891,192.0.2.2,12345,192.0.2.1,60371,47,1,\
  response,1804289383,0,0,1,1.3.6.1.2.1.1.3.0,timeticks,26842224
```

5. Security Considerations

SNMP traffic traces usually contain sensitive information. It is
therefore necessary to (a) remove unwanted information and (b) to
anonymize the remaining necessary information before traces are made
available for analysis. It is recommended to encrypt traces when
they are archived.

Implementations that generate CSV or XML traces from raw pcap files
should have an option to suppress or anonymize values. Note that
instance identifiers of tables also include values, and it might
therefore be necessary to suppress or anonymize (parts of) the

instance identifiers. Similarly, the packet and message headers
typically contain sensitive information about the source and
destination of SNMP messages as well as authentication information
(community strings or user names).

Anonymization techniques can be applied to keep information in traces
that could otherwise reveal sensitive information. When using
anonymization, values should only be kept when the underlying data
type is known and an appropriate anonymization transformation is
available (filter-in principle). For values appearing in instance
identifiers, it is usually desirable to maintain the lexicographic
order. Special anonymization transformations that preserve this
property have been developed, although their anonymization strength
is usually reduced compared to transformations that do not preserve
lexicographic ordering [HS06].

The meta data associated with traces and in particular information
about the organization owning a network and the description of the
measurement point in the network topology where a trace was collected
may be misused to decide/pinpoint where and how to attack a network.
Meta data therefore needs to be properly protected.

6. IANA Considerations

   Per this document, IANA has registered a URI for the SNMP XML trace
   format namespace in the IETF XML registry [RFC3688]. Following the
   format in RFC 3688, the following registration has been made:

   URI: "urn:ietf:params:xml:ns:snmp-trace-1.0"

   Registrant Contact: The NMRG of the IRTF.

   XML: N/A, the URI is an XML namespace.

7. Acknowledgements

   This document was influenced by discussions within the Network
   Management Research Group (NMRG). Special thanks to Remco van de
   Meent for writing the initial Perl script that lead to the
   development of the snmpdump software package and Matus Harvan for his
   work on lexicographic order preserving anonymization transformations.
   Aiko Pras contributed ideas to Section 3 while David Harrington
   helped to improve the readability of this document.

   Last call reviews have been received from Bert Wijnen, Aiko Pras,
   Frank Strauss, Remco van de Meent, Giorgio Nunzi, Wes Hardacker, Liam
   Fallon, Sharon Chisholm, David Perkins, Deep Medhi, Randy Bush, David
   Harrington, Dan Romascanu, Luca Deri, and Marc Burgess. Karen R.

8. References

8.1. Normative References

   [RFC2578]   McCloghrie, K., Perkins, D., and J. Schoenwaelder,
                "Structure of Management Information Version 2 (SMIv2)",
                STD 58, RFC 2578, April 1999.

   [OASISRNG]   Clark, J. and M. Makoto, "RELAX NG Specification",
                OASIS Committee Specification, December 2001.

   [OASISRNC]   Clark, J., "RELAX NG Compact Syntax", OASIS Committee
                Specification, November 2002.

   [RFC3584]   Frye, R., Levi, D., Routhier, S., and B. Wijnen,
                "Coexistence between Version 1, Version 2, and Version 3
                of the Internet-standard Network Management Framework",
                BCP 74, RFC 3584, August 2003.

   [RFC3688]   Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688,
                January 2004.

8.2. Informative References

   [RFC1052]   Cerf, V., "IAB Recommendations for the development of
                Internet network management standards", RFC 1052,
                April 1998.

   [RFC2579]   McCloghrie, K., Perkins, D., and J. Schoenwaelder,
                "Textual Conventions for SMIv2", STD 58, RFC 2579,
                April 1999.

   [RFC3418]   Presuhn, R., Ed., "Management Information Base (MIB) for
                the Simple Network Management Protocol (SNMP)", STD 62,
                RFC 3418, December 2002.

   [RFC2863]   McCloghrie, K. and F. Kastenholz, "The Interfaces Group
                MIB", RFC 2863, June 2000.

   [RFC3410]   Case, J., Mundy, R., Partain, D., and B. Stewart,
                "Introduction and Applicability Statements for Internet-
                Standard Management Framework", RFC 3410, December 2002.

   [RFC4022]   Raghunarayan, R., "Management Information Base for the
                Transmission Control Protocol (TCP)", RFC 4022,
                March 2005.

   [PDMQ04]   Pras, A., Drevers, T., van de Meent, R., and D. Quartel,
                "Comparing the Performance of SNMP and Web Services based
                Management", IEEE Transactions on Network and Service
                Management 1(2), November 2004.

   [Pat01]   Pattinson, C., "A Study of the Behaviour of the Simple
                Network Management Protocol", Proc. 12th IFIP/IEEE
                Workshop on Distributed Systems: Operations and
                Management , October 2001.

   [DSR01]   Du, X., Shayman, M., and M. Rozenblit, "Implementation
                and Performance Analysis of SNMP on a TLS/TCP Base",
                Proc. 7th IFIP/IEEE International Symposium on Integrated
                Network Management , May 2001.

   [CT04]   Corrente, A. and L. Tura, "Security Performance Analysis
                of SNMPv3 with Respect to SNMPv2c", Proc. 2004 IEEE/IFIP
                Network Operations and Management Symposium , April 2004.

   [PFGL04]   Pavlou, G., Flegkas, P., Gouveris, S., and A. Liotta, "On
                Management Technologies and the Potential of Web
                Services", IEEE Communications Magazine 42(7), July 2004.

   [SM99]   Sprenkels, R. and J. Martin-Flatin, "Bulk Transfers of
                MIB Data", Simple Times 7(1), March 1999.

   [Mal02]   Malowidzki, M., "GetBulk Worth Fixing", Simple
                Times 10(1), December 2002.

   [HS06]   Harvan, M. and J. Schoenwaelder, "Prefix- and
                Lexicographical-order-preserving IP Address
                Anonymization", IEEE/IFIP Network Operations and
                Management Symposium NOMS 2006, April 2006.

   [XFA02]   Xu, J., Fan, J., and M. Ammar, "Prefix-Preserving IP
                Address Anonymization: Measurement-based Security
                Evaluation and a New Cryptography-based Scheme", 10th
                IEEE International Conference on Network
                Protocols ICNP'02, November 2002.

   [FXAM04]   Fan, J., Xu, J., Ammar, M., and S. Moon, "Prefix-
               Preserving IP Address Anonymization", Computer
               Networks 46(2), October 2004.

   [PAPL06]   Pang, R., Allman, M., Paxson, V., and J. Lee, "The Devil
               and Packet Trace Anonymization", Computer Communication
               Review 36(1), January 2006.

   [RW07]   Ramaswamy, R. and T. Wolf, "High-Speed Prefix-Preserving
               IP Address Anonymization for Passive Measurement
               Systems", IEEE Transactions on Networking 15(1),
               February 2007.

URIs

   [1] <http://en.wikipedia.org/wiki/Pcap>

   [2] <http://www.tcpdump.org/>

   [3] <http://www.wireshark.org/>

   [4] <http://www.datcat.org/>

   [5] <https://svn.eecs.jacobs-university.de/svn/schoenw/src/snmpdump>

   [6] <http://xmlsoft.org/XSLT/>

   [7] <http://perl-xml.sourceforge.net/faq/>

   [8] <http://www.relaxng.org/>

Author's Address

   Juergen Schoenwaelder
   Jacobs University Bremen
   Campus Ring 1
   28725 Bremen
   Germany

   Phone: +49 421 200-3587
   EMail: j.schoenwaelder@jacobs-university.de