University of Amsterdam

University of Amsterdam
System and Network Engineering

Research Project 1 - Monitoring DNSSEC

Authors:
Martin Leucht
Martin.Leucht@os3.nl

Julien Nyczak
Julien.Nyczak@os3.nl

Supervisor:
Rick van Rein

February 8, 2015

Abstract

Monitoring DNSSEC zones is essential to keep them fully operational. Although SNMP is a very old fashioned protocol, it is still widely used to monitor devices on IP Networks and can be implemented for applications as well, such as DNSSEC. This paper focuses on building a proof of concept for monitoring DNSSEC through SNMP. To that aim, we have constructed an SNMP MIB module that, once loaded to an SNMP subagent, provides DNSSEC critical parameters to a monitoring station. DNSSEC data is retrieved periodically by Python scripts and inserted into a central XML file parsed by the SNMP subagent. New zones are handled automatically when simply specified in a configuration file. The usage of SNMP makes our proof of concept easily deployable. However, it lacks features such as notifications and might show performance issues when it comes to monitor a large number of zones.

Contents

1 Introduction

As figures demonstrate, DNSSEC (Domain Name System Security Extension) becomes more and more popular. Indeed, 77% of top-level domains (TLDs) have trust anchors published as delegation signer (DS) records in the root zone[1]. However, to function properly, resource records (RR) of a DNSSEC-enabled zone need to be regularly signed since signatures have an expiration date. If signatures have not been refreshed in time, the zone becomes unreachable. This is not acceptable.
Monitoring DNSSEC parameters, such as signature expiration dates, could avoid those undesired effects. For instance, a monitoring system would send notifications to the zone maintainer when a resource record signature (RRSIG) of a resource record is about to expire.
There exist a lot of approaches to monitor DNSSEC. For example, service providers offering DNSSEC services have developed their own utilities, often strongly related to their infrastructures and demands. None of these solutions uses the Simple Network Management Protocol (SNMP). SNMP provides an abstraction model for data required to be monitored. This data is represented in a standard format, known as an SNMP Management Information Base (MIB). Thus, it makes it applicable for a generic use in monitoring systems, rather than for custom-made solutions. The absence of a MIB module for DNSSEC urges to develop such a solution.

1.1 Related Work

As pointed out earlier, there are plenty of monitoring systems already existing for DNSSEC[2], but we are looking for an integrated SNMP-compliant tool. We want to write our SNMP sub-agent in Python, and some work has already been done towards that direction. Indeed, *python-netsnmpagent*[3] is a Python module that facilitates the writing of Net-SNMP subagents in Python. Furthermore, two DNS server MIB[4] [5] modules have already been created. Their purpose was to manage DNS name servers (e.g. updating zones) but have been retired since[6] because inter alia, goals were not clearly defined. However, MIB modules for DNSSEC monitoring do not seem to have ever been created.

1.2 Research Questions

Our main goal is to develop an SNMP-compliant monitoring proof of concept. This implies the definition of a set of variables meaningful to the well-being of a DNSSEC signed zone, the construction of a MIB module for DNSSEC, and the development of an SNMP agent meant to feed the MIB. This naturally leads to the following research questions.

- What are vital life signs for monitoring DNSSEC?
- How to construct a MIB module for DNSSEC?
- How to conduct monitoring based on such a MIB?
- How do architectures for monitoring DNSSEC compare?

2 Background

2.1 SNMP

SNMP is an IETF (Internet Engineering Task Force) Internet Standard to manage and monitor devices running on IP networks and is defined in several RFC's [7]. SNMP is a connectionless protocol using UDP Port 161. SNMP provides an abstract interface to serve status information about IP capable devices or applications to a management station. Classical candidates that support SNMP are network devices such as routers and switches. SNMP can also be implemented on Operating Systems or application extensions.
SNMP provides an agent-manager architecture. The agent facilitates an SNMP interface that allows a manager to get (or sometimes set) management information data from a managed device in an SNMP-specific form. The "set" facilities never made it to real-life use because of its limited security and that is also the main reason why SNMP is only used in "get" mode. Other protocols seem to be leading the set aspect. For example one could use IPsec, or might send SNMP over SCTP to have an additional protection layer to circumvent the insecurity of the SNMP protocol. Version 3 of the SNMP protocol includes authentication, privacy, and access control [11].
An agent can be implemented as a subagent that registers data to a master agent by using the AgentX protocol [8]. The structure of the management information and available SNMP variables are defined in a Management Information Base (MIB) which follows strict syntax rules defined in the Structure of Management Information Version 2 (SMIv2) [9]. SMIv2 is a subset of ASN.1.
SNMP variables are assigned to Object Identifiers (OID) and organized in a hierarchical structure. Before writing an SNMP MIB, an OID entry point inside the global MIB tree has to be defined. Those entry points are located under the internet branch [12]. Public branches inside the *mgmt* subtree are defined by the IETF. Private branches inside the *private* subtree are assigned to companies by the Internet Assigned Numbers Authority (IANA). An OID requires to be unique to avoid overlapping with other MIBs. An *experimental* subtree exists for testing purposes.

2.2 DNSSEC

DNS stands for Domain Name System. It is a decentralized hierarchical distributed system that mainly translates domain names to IP addresses. As DNS does not provide integrity nor authentication, DNS data can be subject to forgery. Hence, DNSSEC has been specified to solve this problem. It uses public key cryptography and cryptographically signs the resource records of a zone so a chain of trust can be build from the root down. To understand the concept of this chain of trust, let's walk it from bottom up, with *derby.os3.nl.* as a zone example.
First of all, the *derby.os3.nl.* zone must have at least two (for key management simplification) key-pairs published in the DNS, a Key Signing Key (KSK) and a Zone Signing Key (ZSK). They form the DNSKEY RRset of the zone. The ZSK is used to sign all the resource records in a zone. It is usually 1024 bits long and is roll-over more often than a KSK (every two weeks is common practise). The main reason for key-rollovers is that the longer a key is published, the higher is the probability that the key gets compromised by an attacker. A KSK is meant to validate the DNSKEY RRset. Since it is supposed to be roll-over less frequently than a ZSK, it is obvious that its size is bigger (most often 2048 bits). In addition, the KSK needs to be validated as well. To that end, the hash of the public key is published in the parent zone (*os3.nl.* in our example) as a DS record. In other words, the KSK acts a Secure Entry Point (SEP) from the zone *os3.nl.* to *derby.os3.nl.*. Next, the same validation process occurs for the *os3.nl.* zone, then for the *nl.* zone to finally reach the root zone. Any DNSSEC-capable resolver knows about the public portion of the root zone's KSK published by ICANN. This key acts as the trust anchor for DNSSEC resolvers.
This chain can be broken if one of its link does not behave as it should. This is the reason why monitoring some of the elements mentioned above is essential for a DNSSEC zone's sake.

3 Approach and Methods

3.1 Overview

Developing an SNMP based prototype to monitor DNSSEC relevant data requires to implement several components that collaborate with each other. During the implementation of those component, one focus was to use programming libraries to get DNSSEC related data, rather than using existing external tools (e.g. dig [17] or validns [18]). That makes the prototype as independent as possible and thus, it enables simple integration in existing infrastructures.
The main component of the prototype is the SNMP MIB module that has been created to cover DNSSEC critical data. Another important component is the SNMP subagent that has been partly developed during the research project. The underlying SNMP implementation for the subagent is based on the NET-SNMP toolkit [14]. It includes applications (snmpget, snmpwalk, etc) to retrieve SNMP data from an agent. Moreover it provides libraries and programming language APIs to allow implementing own subagents. The subagent is written in Python and communicates via the AgentX [8] protocol to the NET-SNMP master agent. The decision to implement an AgentX subagent is based on the fact that it simplifies the handling of SNMP protocol specific details, such as SNMP variable registration. The subagent itself is filled in with data by sotftware components that collect information from authoritative name servers serving DNSSEC enabled zones. To be more precise, those software components are data wrapper scripts that make use of the *dnspython* [16] library.

3.2 Vital life signs for DNSSEC

There are many variables that can be monitored to keep a DNSSEC zone up and running properly. Here is a non-exhaustive list of what we think to be the most pertinent ones.

- One of the first variable that comes up to mind is the availability of a zone from a resolver point of view. If the local resolver cannot validate the NS (Name Server) record of a zone, it probably means its signature has expired and leads to a SERVFAIL status. Thus, the administrator can spot easily the unavailability of a zone.

- As we previously mentioned, the KSK signs the DNSKEY RRset and acts as a SEP. It is seems legitimate to verify the signature of the DNSKEY RRset against the published KSK. In doing so, the monitoring solution assumes that the advised setting of the SEP bit on the KSK and only the KSK is followed by the DNSSEC implementation, which is commonly the case.

- In order not to break the chain of trust, there must be at least one match between a DS RR in the parent zone and a DNSKEY RR in the child zone for every signature algorithm (RSA-SHA1, RSA-SHA256, DSA-SHA1, etc). Hence, the number of delegations must be at least equal to the number of DS records on the parent side. Additional DS or DNSKEY RR are not harmful.

- TTL (Time to Live) values are useful to cache replies of previous requests to limit the load on servers. But from a DNSSEC perspective, TTL values should not be set randomly. According to the RFC 4641bis, on one hand the maximum zone TTL of the RRset of a zone should be several times smaller than the validity period of the RRSIGs. On the other hand, the minimum zone TTL value should be long enough to validate the whole chain of trust and to avoid high loads on recursive name servers.

- It is common practise to have a zone served by at least two severs, a master and a slave. While monitoring a zone, it is important to know how many servers deliver it and from which one the data is retrieved from (name and IP address).

- Signatures have an inception and an expiration date. If the RRSIG of an RR expires, the record stops being available. This expiration date is obviously one of the most relevant variable to monitor. Good candidates for this variable are the signatures on the SOA, NS and DNSKEY records as they are key features of the availability of a zone. The oldest signature, that represents an inception date, might also give an indication of the status of a zone, as well as the number of expired signatures. Inception dates of RRSIGs are also worth to be monitored. If such a date is in the future it will also result in the unavailibility of the corresponding RR.

- Discrepancies between zone files on a master and a slave server might occur if AXFR encounters issues. The rule wants the serial number of a zone file on a master server to be increased each time the file is modified. Hence, comparing serial numbers of the same zone delivered by a master and a slave server highlights zone transfer problems. The discrepancy check can be extended to records themselves in order to make it fine-grained.

Once all the variables are known, they can be organised and grouped to form the scalar and tabular objects of the MIB module. The MIB module design is based on these variables.

3.3 SNMP MIB design

Creating an SNMP MIB requires to have an OID entry point as described in Section 2. Moreover several design considerations have to be taken into account. In this section, an overview of design considerations related to our MIB is given. It is not intended to be a description for each detail that a MIB designer needs to obey.
Firstly, one need to figure out what data will go into the MIB and how different data groups are separated in subbranches starting from the assigned OID entry point. Data can be organized in columnar or scalar objects. Scalar objects define a single object instance, whereas columnar objects allow to represent tables.

3.3.1 SNMP table indexing

Tables are the most complex objects that can be defined inside a MIB. Table rows are referenced by indexes comparable to primary keys in database structures. In SNMP, it is common to use the term conceptual tables [13]. Table row entries and instances of them are created or deleted dynamically by an SNMP agent each time a new variable is registered to a row.
In contrast to database structures, indexes for SNMP tables can be assigned to several basic datatypes or combinations of them, like integer or string variables [9].
In our MIB, the primary key of all tables is represented by the domain name of a DNS zone. For this reason datatype OCTET-STRING is used for indexing of all tables. Furthermore, it is more intuitive to handle with strings rather than with long numerical OIDs. However, inside the payload of SNMP packets, when datatype OCTET-STRING is used for indexes, each letter is represented as the corresponding decimal ASCII value. It might be required to go beyond the ASCII character set by using the Unicode character set [1], particularly when Internationalized Domain Names (IDN) [25] are considered to be taken into account. Then strings would be encoded as UTF-8 [27]. One could circumvent that, by first converting domain names that include international characters, using the Punycode [26] algorithm, before the domain is linked as value inside a MIB object. The Punycode algorithm offers the capability to represent Unicode with the limited character subset of ASCII. To provide the ability to cover internationalized domain names without converting them into ASCII characters (by applying the Punycode algorithm), UTF-8 encoding is allowed to represent domain names as indexes and values of instances of objects within the MIB.

3.3.2 Data types and textual conventions

A further point which needs to be considered is the choice of the right datatype for each object-type in the MIB. The SNMP protocol and the underlying ASN.1 types allow three basic datatypes for data fields in an SNMP message [21]:

[1]the unicode character set includes ASCII as a subset

- INTEGER based types (Counters, Gauges, Integers)
- OCTET-STRING based types (Displayable Strings, IpAddress DateAndTime, etc)
- OBJECT IDENTIFIERS

Basic numeric values, e.g. the number of zones covered by the DNSSEC MIB, are represented as simple INTEGER datatypes. OCTET-STRINGS are used to cover string values, for example domain names. To add additional restrictions to values of instances of objects, textual conventions can be defined. Textual conventions are a method for creating new datatype definitions and let MIB designers decribe their properties more precisely.
Encoding on the wire of the assigned datatype values defined in a textual convention is based on the ASN.1 Basic Encoding Rules (BER) [22] [21].
Within a textual convention a DISPLAY-HINT clause can be defined that specifies how the desired output format of an instance of an object should look like. Textual conventions are imported by including other MIB modules [10] or by defining them in the MIB itself.

4 Results

4.1 Data collection

Collecting data from zones is not an easy task, especially when the amount of available time is relatively small. This is the reason why this assignment has been divided in two Python scripts[2], each developed by one of us. The first script has two purposes:

- Creating an XML template *data.xml*
- Collecting and inserting DNSSEC data into a new XML file (*updated.xml*) which is based on *data.xml*

The second script focuses on updating *updated.xml* with other DNSSEC data. It is worth noticing that our proof of concept is still at a beta level and needs to be enhanced. For instance, the scripts written for data retrieval fail if the name server of a zone is down. That said, let's assume they are all up.

4.1.1 General data structure

For security and scalability reasons, the SNMP subagent cannot fetch the data directly from the monitored zones. Indeed, some of the checks the scripts perform are expensive[3] and data retrieval would have became a real burden for the SNMP subagent when dealing with a large number of domains. It must retrieve the data from a central repository instead. *XML* (EXtensible Markup Language) seems to be the appropriate choice as it is meant to represent data-structures in a human and machine readable format, where the information is easily accessible for applications. However, the use of XML might affect performances when dealing with a large number of zones. This has not been witnessed during this project though, as we dealt with four zones only.
We used Python and the *ElementTree* XML API[20] to build an XML template describing for each zone the MIB objects and their syntax. The resulting XML document is shown in listing 1

```
<?xml version="1.0" encoding="UTF-8" ?>
<ZoneList>
  <Zone id="1" name="berlin.warsaw.practicum.os3.nl">
    <table name="dnssecZoneGlobalTable">
      <item>
        <data id="2" name="dnssecZoneGlobalServFail" type="Integer32"> </data>
      </item>
      [...]
    </table>
    [...]
  </Zone>
  [...]
</ZoneList>
```

Listing 1: XML template

The main element of the file is *ZoneList*, which is divided into *Zone* sub-elements. Each zone element has two attributes which are *id* and *name* defining respectively the zone id and its name. The former is incremented for each new zone element, and the latter is retrieved from a text file[4] that contains zones to monitor. The *Zone* element is split into sub-elements as well, *table*. *table* has one attribute *name* which

[2]The scripts are data_wrapper_JN.py and data_wrapper_ML.py

[3]Verifying the DNSKEY RRset against the published KSK involves the *Crypto* Python module.

[4]The text file "zone_hint" is made of key/value pairs (zone/IP). The hard-coded IP address is used only if the corresponding zone shows a SERVFAIL status.

values are tables defined in the MIB module, namely *dnssecZoneGlobalTable*, *dnssecZoneSigTable*, *dnssec-ZoneAuthNSTable*, *dnssecZoneDiffTable*. Each table is divided into sub-elements *item*. An *item* contains a sub-element *data* with three attributes, *id*, *name* and *type*. These represents the objects of the tables defined in the MIB module. The *id* is the OID number of the corresponding table described in the *name* attribute, and *type* is the syntax of the content between the *data* tags. The content is empty so far as it is going to be filled in with DNSSEC data of monitored zones, retrieved by a later part of the script.

4.1.2 Retrieving DNSSEC data

The XML template is now created and needs to be filled in with DNSSEC data. First of all, we verify whether the zone we want to retrieve data from is available from a resolver point of view in order to make sure that the DNSSEC data can be accessed. If it is not[5], we query its authoritative name server instead. The query consists of either an AXFR or a DNS query made possible thanks to the *dnspython* library that allows to retrieve the data described in section 3.2. Furthermore, a copy of *data.xml*, named *updated.xml*, is created to avoid read/write conflicts. The data freshly retrieved is inserted into it by means of a Python dictionary and XPath queries. The values of each item for each table represents a key in the dictionary and the retrieved data is the corresponding value as shown in Listing 2. Since the keys are constructed in a specific way ("zone_name" + "_" + "table" + "_" + "item_ID"), the XPath queries allow to navigate through the XML document in order to tell the script where to add DNSSEC data at the right place in the *updated.xml* file.

```
{'warsaw.practicum.os3.nl_dnssecZoneSigTable_4': '"20161212121212"',
'derby.practicum.os3.nl_dnssecZoneDiffTable_2': '1',
'warsaw.practicum.os3.nl_dnssecZoneSigTable_3': '"20161212121212"',
'warsaw.practicum.os3.nl_dnssecZoneSigTable_2': '"20150112142419"',
[...]}
```

Listing 2: Python dictionary

Moreover, we have specified integer values that the scripts return for some checks. They are defined as textual conventions in the MIB and are associated to a specific state as described in table 1. For instance, one could say that according to Listing 2, the zone file for the *derby.practicum.os3.nl* zone is the same on the master and on slave name servers as serial numbers do not differ[6].

4.2 DNSSEC SNMP MIB implementation

4.2.1 General overview

The OID entry point for the DNSSEC MIB is located inside the ARPA2 OID tree (enterprise OID 44469). The MIB module name is ARPA2-Experimental-DNSSEC-MIBv1 and is associated to the numerical OID *.1.3.6.1.4.1.44469.666.53.46.161.1*. The path from the root down to that module is shown in Figure 1.

[5]The zone shows a SERVFAIL status probably because the signature of at least one of its apex RR is expired.

[6]*dnssecZoneDiffTable_2* represents the second item of the *dnssecZoneDiffTable* which checks for discrepancies in zone file serial numbers between the master and the slave name servers.

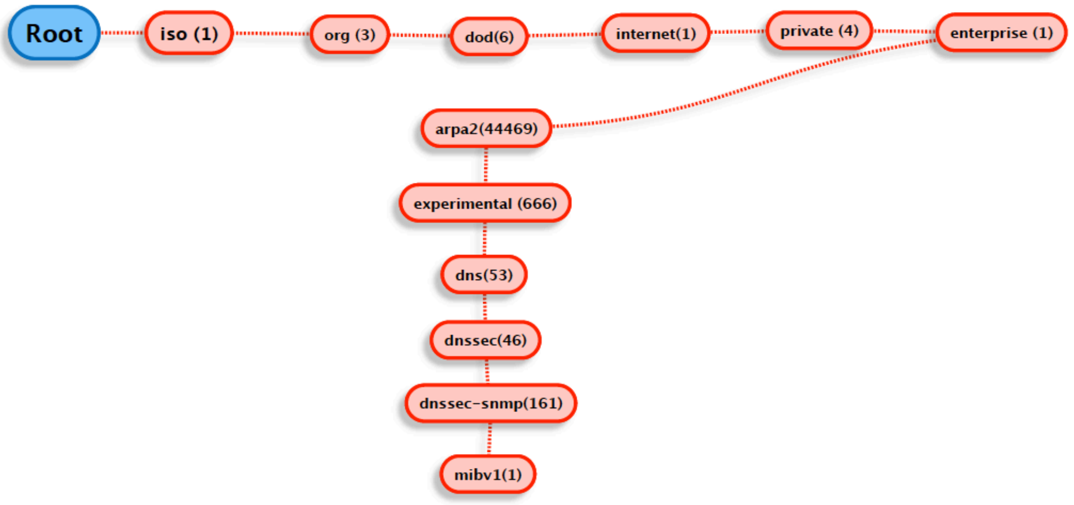


Figure 1: OID Tree

Objects defined in the MIB are organized in tables [7] or scalar types. INTEGER datatypes are used to represent boolean and numeric values and data type OCTET-STRING to represent strings (e.g domain names). Four tables are included in the MIB *(dnssecZoneGlobalTable, dnssecZoneAuthNSTable, dnssecZoneSigTable, dnssecZoneDiffTable)*. To provide an easy mechanism to associate data entities to a specific zone, all MIB entries are indexed by the domain name itself (datatype OCTET-STRING).

4.2.2 Data covered by the MIB

In section 3.2 are listed the most relevant parameters that need to be covered by the MIB, in order to keep track of vital life signs of DNSSEC signed zones.
The first table *(dnssecZoneGlobalTable)* contains general parameters like the availability of the zone seen from the prospect of an external DNSSEC-aware resolver or the status of the signature verification of the DNSKEY RRset against the published KSK. That table also provides general data like the FQDN of the nameserver, where data is fetched from and its IP address. Furthermore entries in that table supply information about the general DNS setup for a zone. For instance, one could retrieve the information if delegations are present in a zone and subsequently check if the count of DS records matches at least the amount of delegations.
Entries in table *dnssecZoneGlobalTable* are not supposed to list all data that is provided by the zone. For example, there is no need to list TTL values for each RR when the maximum and minimum TTL observed in a zone is provided.
The same applies to entries of the *dnssecZoneSigTable*. Only expiration dates of essential RRs at the zone apex are covered, rather than listing each signature expiration date for every RR. For the same reason, entries in the *dnssecZoneDiffTable* only provide a general indication if discrepancies between data provided by a master and slave exist. The perfect candidate to illustrate differences in a master-slave setup is the serial number of the SOA RR itself. The information if a master-slave setup exist for a particular zone can be retrieved from the *dnssecZoneGlobalAuthNSCount* entry in the *dnssecZoneGlobalTable*.
The general idea of the data provided in all tables in the MIB is to reflect the health status of a zone in a way that it can be easily retrieved from monitoring tools without the need of complex computations.
Listing **??** shows all top level elements of the created MIB including tables and their associated indexes.

[7] In SNMP terminology, tables are denoted as conceptual tables or columnar objects

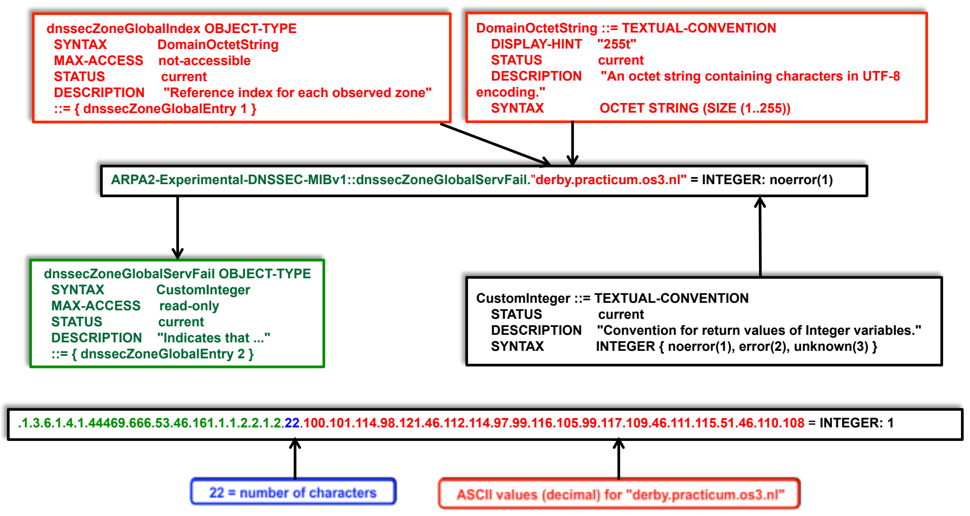
```
+--arpa2experimentaldnssecMIBv1(1)
   |
   +--dnssecObjects(1)
   |  |
   |  +--dnssecGeneral(1)
   |  |  |
   |  +--dnssecZoneGlobal(2)
   |  |  |
   |  |  +--dnssecZoneGlobalTable(2)
   |  |     |
   |  |     +--dnssecZoneGlobalEntry(1)
   |  |        |  Index: dnssecZoneGlobalIndex
   |  |
   |  +--dnssecZoneAuthNS(3)
   |  |  |
   |  |  +--dnssecZoneAuthNSTable(3)
   |  |     |
   |  |     +--dnssecZoneAuthNSEntry(1)
   |  |        |  Index: dnssecZoneGlobalIndex
   |  |        |
   |  +--dnssecZoneSig(4)
   |  |  |
   |  |  +--dnssecZoneSigTable(4)
   |  |     |
   |  |     +--dnssecZoneSigEntry(1)
   |  |        |  Index: dnssecZoneGlobalIndex
   |  |        |
   |  +--dnssecZoneDiff(5)
   |     |
   |     +--dnssecZoneDiffTable(5)
   |        |
   |        +--dnssecZoneDiffEntry(1)
   |           |  Index: dnssecZoneGlobalIndex
   |           |
   +--dnssecMIBConformance(2)
      |
      +--dnssecMIBGroups(1)
      |  |
      |  +--dnssecMIBScalarGroup(1)
      |  +--dnssecMIBTableGroup(2)
      |
      +--dnssecMIBCompliances(2)
```

Listing 3: Structure of ARPA2-Experimental-DNSSEC-MIBv1

4.2.3 Usage of textual conventions and table indexing

To add more restrictions to the defined object-types and their indexes, textual conventions are used. Figure 2 shows the impact of textual conventions for object-type *dnssecZoneGlobalServFail* in table *dnssecZone-GlobalTable* and its representation inside an SNMP packet.

A textual convention *DomainOctetString* is defined and mapped to the *dnssecZoneGlobalIndex*. That textual convention specifies the usage of the underlying datatype OCTET STRING and limits its number of octets to 255. That number represents the maximum allowed number of octets a domain name can con-



```
dnssecZoneGlobalIndex OBJECT-TYPE
  SYNTAX          DomainOctetString
  MAX-ACCESS      not-accessible
  STATUS          current
  DESCRIPTION     "Reference index for each observed zone"
  ::= { dnssecZoneGlobalEntry 1 }
```

```
DomainOctetString ::= TEXTUAL-CONVENTION
   DISPLAY-HINT    "255t"
   STATUS          current
   DESCRIPTION     "An octet string containing characters in UTF-8
encoding."
   SYNTAX          OCTET STRING (SIZE (1..255))
```

```
ARPA2-Experimental-DNSSEC-MIBv1::dnssecZoneGlobalServFail."derby.practicum.os3.nl" = INTEGER: noerror(1)
```

```
dnssecZoneGlobalServFail OBJECT-TYPE
  SYNTAX          CustomInteger
  MAX-ACCESS      read-only
  STATUS          current
  DESCRIPTION     "Indicates that ..."
  ::= { dnssecZoneGlobalEntry 2 }
```

```
CustomInteger ::= TEXTUAL-CONVENTION
   STATUS          current
   DESCRIPTION     "Convention for return values of Integer variables."
   SYNTAX          INTEGER { noerror(1), error(2), unknown(3) }
```

```
.1.3.6.1.4.1.44469.666.53.46.161.1.1.2.2.1.2.22.100.101.114.98.121.46.112.114.97.99.116.105.99.117.109.46.111.115.51.46.110.108 = INTEGER: 1
```

22 = number of characters
ASCII values (decimal) for "derby.practicum.os3.nl"

Figure 2: Effect of textual conventions to represent MIB data

tain [23]. Although the maximum allowed number of octets a domain name can contain is limited to 255, an IDNE proposal [28] (Internationalized domain names using EDNS) exists that uses the DNS extension mechanism called EDNS [29]. The extension allows some IDNE labels to be longer than 63 characters and some IDNE names to be longer than 255 octets. That would allow to send domain names either as ASCII or binary, and the binary format is UTF-8. The *DISPLAY-HINT* clause specifies the usage of UTF-8 encoded characters [10]. That string, which represents a domain name is appended as an index value to each instance of an object-type.
The textual convention *CustomInteger* restricts the values that object-type *dnssecZoneGlobalServFail* can have. In that case we allow the values shown in table 1. That enables us to implement our agent data wrapper scripts in a way that for each check only these values can occur and we can associate them to a textual equivalent. Figure 2 also shows the numerical representation of an instance of object-type *dnssecZoneGlobalServFail* and how the OID for the SNMP message is constructed on the wire. The green marked numbers are the numerical identifiers for the object-type itself. Then the number of characters of the associated index is appended to the OID and finally the ASCII values in decimal for each character of the domain name.

Value	Meaning
1	No error
2	Error
3	Unknown

Table 1: Textual convention for integer variables

4.3 SNMP Subagent implementation

The SNMP subagent itself is based on the work of Pieter Hollants [3] and provides us a Python API to communicate with the NET-SNMP master agent. The main function that updates the defined SNMP objects was implemented during the research project. That function uses the API calls shown in Table 2.

API call	Description
`tablename.clear`	clears a table row
`tablename.addRow`	adds a table row
`tablename.rowid.setRowCell`	sets a value to a cell in a table row

Table 2: API calls python-netsnmpagent

The values that are assigned to object-type instances are derived from an XML data file. The function that implements the SNMP variables-update reads in the XML structure and iterates for all announced zones through the XML document using XPath queries [24]. Everytime a corresponding table entry is identified, that entry, and in turn the related SNMP variable get updated. The function is included in the *agent* directory on Github [32]. Figure 3 shows the workflow of the function.

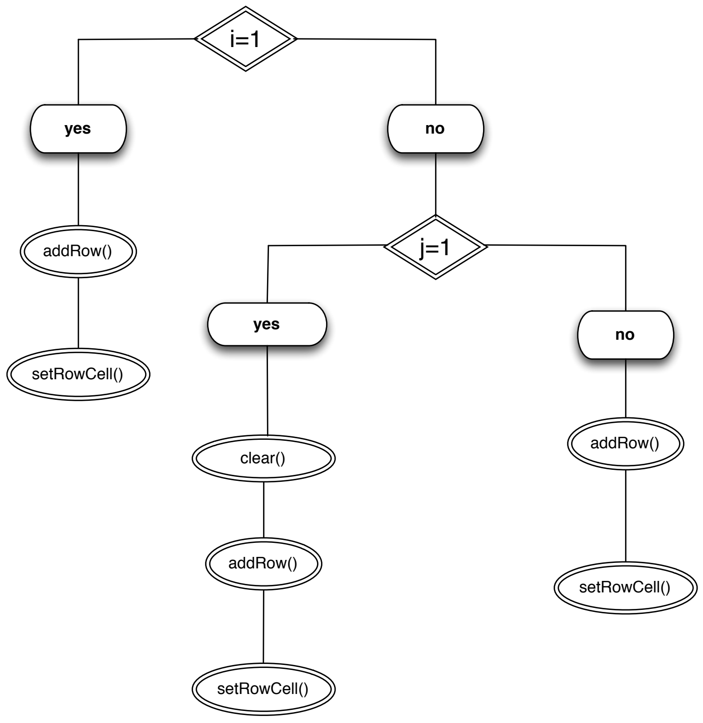


Figure 3: workflow of the subagent's UpdateSNMPObjs() function

The function first checks if it is executed the first time *(i=1)*. If so, table rows are added instantly. The counter variable i is incremented each time the function is called. If i is not equal to one, another counter variable j is checked. That counter indicates which round of iteration is currently performed inside the function. Only, and only if it is the first iteration, the current table is cleared and after that new values are added.
The SNMP subagent operates asynchronously. The data update thread is decoupled from the data providing thread. This ensures periodic data updates and also makes sure that SNMP requests will always be answered to in time. Zones can be added or removed dynamically by specifying them in the *zone_hint* file. Figure 4 shows the interaction between all components that are involved.

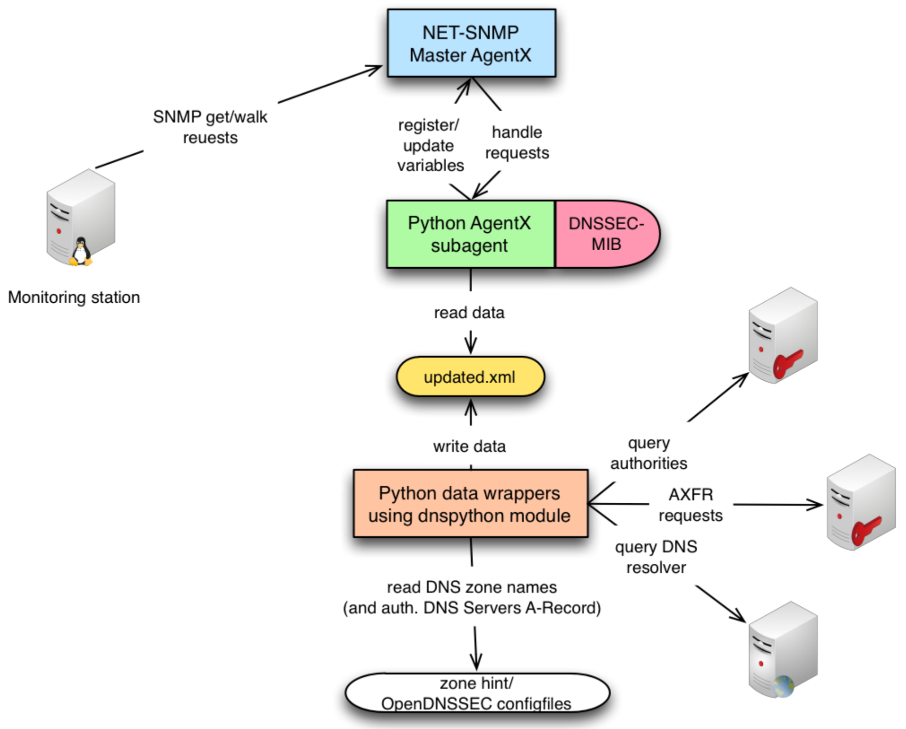
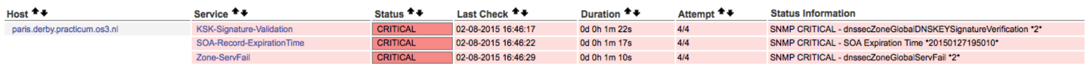


Figure 4: logical scheme SNMP subagent

4.4 Conduct DNSSEC monitoring

In order to present how monitoring can be conducted using our created MIB, a use-case example is sketched. To implement the monitoring tasks, the well known monitoring system Nagios [30] and its SNMP plugin *check-snmp* [31] is used.
Let's assume the RRSIG RR of the SOA record and the DNSKEY RR (KSK) of the DNSSEC signed zone *paris.derby.practicum.os3.nl* have expired. That will lead a DNSSEC-aware resolver to invalidate the complete zone. A DNSSEC-aware resolver will only return a SERVFAIL without specifying the root cause for the invalidation when it is queried for RR of that zone. In that case, querying the corresponding OIDs using our MIB might be helpful. Values for the object-type *dnssecZoneGlobalServFail* will return an error (2) but also the validation check *dnssecZoneGlobalDNSKEYSignatureVerification* in table *dnssecZoneGlobalTable* will return an error (2) for that zone.
The actual reason for the SERVFAIL can be derived from table *dnssecZoneSigTable*, in particular from object-type *dnssecZoneSigDNSKEYSignatureExpirationTime*. It will show an expired date for that RRSIG RR, in that case the expiration of the RRSIG occurred on 27th of January 2015 at 19:50:10. Figure 5 shows the applied Nagios implementation.



Host	Service	Status	Last Check	Duration	Attempt	Status Information
paris.derby.practicum.os3.nl	KSK-Signature-Validation	CRITICAL	02-08-2015 16:46:17	0d 0h 1m 22s	4/4	SNMP CRITICAL - dnssecZoneGlobalDNSKEYSignatureVerification *2*
	SOA-Record-ExpirationTime	CRITICAL	02-08-2015 16:46:22	0d 0h 1m 17s	4/4	SNMP CRITICAL - SOA Expiration Time *20150127195010*
	Zone-ServFail	CRITICAL	02-08-2015 16:46:29	0d 0h 1m 10s	4/4	SNMP CRITICAL - dnssecZoneGlobalServFail *2*

Figure 5: implemented Nagios checks based on values of the DNSSEC-MIB

5 Conclusion

Our research project has been divided into several steps in order to build a proof of concept that can be implemented to monitor DNSSEC through SNMP. First of all, we have defined what are the vital life signs of a DNSSEC zone, i.e. the variables that are essential to the proper functioning of such a zone. Then, we have constructed a Management Information Base module for SNMP, located under the ARPA2 OID tree, that delivers the variables previously defined for each zone that is monitored. In addition, we have written an SNMP subagent in Python based on the *python-netsnmpagent* module, that communicates with the NET-SNMP master agent and updates the SNMP objects. We also have written Python scripts that collect DNSSEC data for each monitored zone into a central XML file easily accessible by the SNMP subagent. A new zone can be monitored automatically just by adding its domain name with its corresponding name server IP address in the *zone_hint* file. Finally, we have conducted monitoring based on our proof of concept using the Nagios monitoring system. Other monitoring architectures for DNSSEC employ different ways to retrieve DNSSEC data such as remote procedure calls. As SNMP is a standard application protocol, our proof of concept can be deployed easily to monitor DNSSEC enabled zones. However, it has been tested with only four zones, and might face performance issues when dealing with a large number of zones for several reasons, inter alia, the adoption of XML, AXFR requests to retrieve data and expensive checks (e.g. usage of the Python *Crypto* module).
The DNSSEC-MIB file, the subagent and all related software components are available on Github [32].

6 Future work

Due to the lack of time, our proof of concept is unfortunately not complete and can be improved. Indeed, our MIB module could deliver more DNSSEC variables. For instance, we could include:

- The validation of the NSEC/NSEC3 chain.
- The validation of all the RRSIGs against the published ZSK and return the number of invalidated RRSIGs.
- The validation of the DS record in the parent zone against the published KSK of the corresponding child zone.

Moreover, as we do not use the Punycode algorithm, international domain names are not handled by the MIB module. In addition, notifications (so-called SNMP trap) are not implemented. This feature is not available in the *python-netsnmpagent* module yet and the amount of time allowed for the project was too short for us to develop it. Finally, our Python scripts for retrieving DNSSEC data lack of exception handling.

Acknowledgments

We would especially like to express our gratitude to our supervisor Rick van Rein for his support and feedback which led to the success of this project. We would also like to acknowledge Roland van Rijswijk for his insights on the DNSSEC monitoring topic. Finally, we would like to thank the System and Network Engineering master who provided us the required equipment for this project.

References

[1] ICANN, *TLD statistics regarding DNSSEC*, January 2014, http://stats.research.icann.org/dns/tld_report/

[2] Dan York, *DNSSEC tools*, January 2012, http://www.internetsociety.org/deploy360/dnssec/tools/

[3] Pieter Hollants, *Python-netsnmpagent Module*, 2013, https://github.com/pief/python-netsnmpagent

[4] R. Austein & J. Saperia, *DNS Server MIB Extensions*, May 1994, https://tools.ietf.org/html/rfc1611

[5] R. Austein & J. Saperia, *DNS Server MIB Extensions*, May 1994, https://tools.ietf.org/html/rfc1612

[6] R. Austein, *Applicability Statement for DNS MIB Extensions*, November 2001, https://tools.ietf.org/html/rfc3197

[7] SNMP RFC, *Overview of SNMP related RFC's*, 2013 http://www.snmp.com/protocol/snmp_rfcs.shtml

[8] RFC2741, *Agent Extensibility (AgentX) Protocol*, January 2000, https://www.ietf.org/rfc/rfc2741.txt

[9] RFC2578, *Structure of Management Information Version 2 (SMIv2)*, April 1999, https://tools.ietf.org/html/rfc2578

[10] RFC2579, *Textual Conventions for SMIv2*, April 1999, https://tools.ietf.org/html/rfc2579

[11] William Stallings (Cisco Systems, Inc), *Security Comes to SNMP: The New SNMPv3 Proposed Internet Standards*, January 2015, http://www.cisco.com/web/about/ac123/ac147/archived_issues/ipj_1-3/snmpv3.html

[12] OID repository, *iso(1) identified-organization(3) dod(6) internet(1)*, October 2013, http://oid-info.com/get/1.3.6.1

[13] T. Perkins, *Understanding SNMP MIBs Revision 1.1.7* ,September 1993, http://www.dsc.ufcg.edu.br/~jacques/cursos/gr/recursos/outros/perkins.pdf

[14] NET-SNMP, *Suite of applications used to implement SNMP*, February 2013, http://www.net-snmp.org/

[15] NET-SNMP, *AgentX extension for NET-SNMP*, May 2011, http://www.net-snmp.org/docs/README.agentx.html

[16] dnspython, *DNS toolkit for Python*, September 2013, http://www.dnspython.org/

[17] dig, *DNS lookup utility and part of the BIND domain name server software suite*, unknown, ftp://ftp.isc.org/isc/bind9/cur/9.10/doc/arm/man.dig.html

[18] validns, *DNS/DNSSEC zone validator*, 2012, http://www.validns.net/

[19] D. Eastlake & C. Kaufman, *First DNSSEC specifications*, January 1997, http://www.ietf.org/rfc/rfc2065.txt

[20] Python Software Foundation, *The ElementTree XML API*, 01 January 2015, https://docs.python.org/2/library/xml.etree.elementtree.html

[21] Rane Corporation, *SNMP: Simple? Network Management Protocol*, December 2005, `http://www.rane.com/note161.html`

[22] ITU-T Recommendation X.690 , *ASN.1 encoding rules*, July 2002, `http://www.itu.int/ITU-T/studygroups/com17/languages/X.690-0207.pdf`

[23] Wikipedia, *Domain Name System*, February 2015, `http://en.wikipedia.org/wiki9/Domain_Name_System#Domain_name_syntax`

[24] Python Software Foundation, *XPath Support*, January 2015, `https://docs.python.org/2/library/xml.etree.elementtree.html#elementtree-xpath`

[25] RFC 5891 *Internationalized Domain Names in Applications (IDNA)*, August 2010, `https://tools.ietf.org/html/rfc5891`

[26] RFC 3492, *Punycode: A Bootstring encoding of Unicode for Internationalized Domain Names in Applications (IDNA)*, March 2003, `hhttps://www.ietf.org/rfc/rfc3492.txt`

[27] RFC 3629, *UTF-8, a transformation format of ISO 10646*, November 2003, `https://tools.ietf.org/html/rfc3629`

[28] draft-ietf-idn-idne-00, *Internationalized domain names using EDNS (IDNE)*, July 2000, `https://tools.ietf.org/html/draft-ietf-idn-idne-00`

[29] RFC 6891, *Extension Mechanisms for DNS (EDNS(0))*, April 2013, `https://tools.ietf.org/html/rfc6891`

[30] Nagios, *Nagios - The Industry Standard In IT Infrastructure Monitoring*, February 2015, `http://www.nagios.org/`

[31] check-snmp, *The check-snmp Plugin* , 2014, `https://nagios-plugins.org/doc/man/check_snmp.html`

[32] Software components and SNMP MIB, *DNSSEC-SNMP-MIB, SNMP subagent and components*, February 2015, `https://github.com/arpa2/dnssec-snmp-mib`