OpenDNSSEC Enforcer

Use Cases, design decisions and other stuff

Table of Contents

1 Introduction....2
2 Terminology....2
3 Technology....2
4 Testing....2
5 Building....3
6 Install....3
7 Architecture....3
8 Work to be done....3
9 Data Model....3
  9.1 ER....4
  9.2 KASP....5
10 Use cases (sort of)....6
  10.1 Starting the Enforcer....6
  10.2 Stopping the Enforcer....6
  10.3 Processing at startup....7
  10.4 Creation of a policy....7
  10.5 Setting Parameter Values....7
  10.6 Addition of a new Adapter....7
  10.7 Addition of a zone to a policy....7
  10.8 Addition of an HSM to the system....8
  10.9 Creating new Parameters....8
  10.10 Creating new Categories....8
  10.11 Key creation....8
  10.12 Key Rollover....10
  10.13 General Running....10
11 Communication with Signer Engine....10
12 Logging....11
13 Error Handling....11

1 Introduction

This document describes the OpenDNSSEC Enforcer, how it will be used and the logic that it will follow to perform requested and automatic operations.

Occasionally in this document it is assumed that the Enforcer will be running on a Linux like OS. If running on other OS's then appropriate changes my be needed.

2 Terminology

Term	Description
KASP	Key and Signing Policies. A set of policies.
Policy	A series of parameters that will control key rollover and the signing of zones
libksm	An API for manipulating and performing calculations using the policies held in KASP
Enforcer	A daemon that reads the policy and performs the calculations needed to determine when keys are rolled . Also, determines what keys should be in the zone, and what keys should be used to sign the zone. It passes this information to the signer.
Signer Engine	Signs the zone using the key information obtained from the enforcer. In the absence of such information, it will sign the zone using the last set of information received. The enforcer and signer operate independently of one another

Some suggestions for new/replacement terms came out of the conversation that we had.

- NSCF (Next Secure Chain Forger) new name for nsecifier
- Integrity Checker new name for KASP Auditor (This is because the integrity checker should not access KASP, but instead should use a separate configuration perhaps with minimum reasonable values).

3 Technology

The code will be written in ANSI C.

For the database both mysql and sqlite3 will be supported in the first version according to a configure flag at compile time. This is because database access code is already written for both these. Generic database access code such as ODBC is being considered.

The code will use the PKCS#11 API to access the (soft)HSM. It will be tested with SoftHSM and a SunSCA6000.

During development the code will be regularly tested on OS X, Ubuntu 8.10, Solaris 10 and ???

4 Testing

Unit tests will be written for every function using CUNIT???

The code will regularly be run through Valgrind to check for memory leaks.

The code will be tested with Coverity Prevent

5 Building

Code will be built using automake, autoconf, configure and make.

6 Install

7 Architecture

The Enforcer is built on top of libksm an initial version of which was originally developed by Stephen Morris at Nominet. There is also a command line interface to libksm that will continue to be maintained by Nominet.

libksm is a API that allow operations to be performed on and calculations to be performed with the policy held in KASP. It is intended that all the key rollover logic will be in libksm and so can be shared by both the command line and OpenDNSSEC applications.

8 Work to be done

The initial work to be done is:

- Merge changes that I and Nominet have made to libksm.
- Tidy up libksm source and sort out building by automake etc.
- Publish libksm on OpenDNSSEC svn
- Continue work on libksm

- Alter existing Enforcer code to use the tidied up libksm
- Sort out building by automake etc.
- Publish existing Enforcer code on OpenDNSSEC svn
- Continue work on Enforcer

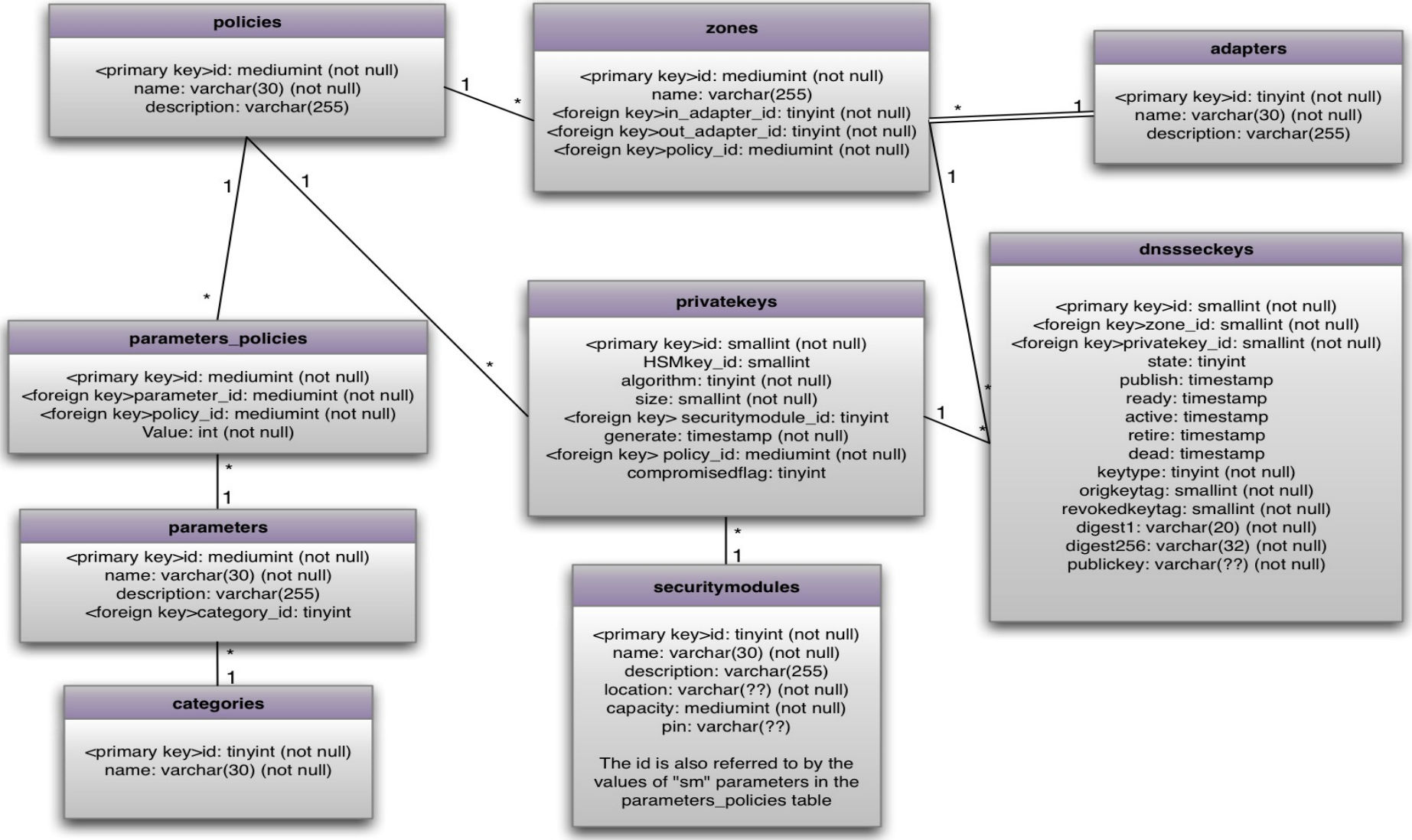
- Alter cli to use tidied up libksm
- Sort out building by automake etc.
- Publish cli on OpenDNSSEC svn
- Continue work on CLI

Details of the further work needed to libksm and the Enforcer will be shown elsewhere.... Further work on the cli is up to Nominet.

9 Data Model

OpenDNSSEC uses policy data stored in KASP and meta-data stored in the meta-store. Both these are stored in a database that will be used by the Enforcer. For simplicity they are stored in the same database schema. The KASP data is stored in the policy and parameters tables. The meta-store is the dnsseckeys table.

9.1 ER


policies
<primary key>id: mediumint (not null)
name: varchar(30) (not null)
description: varchar(255)
zones
<primary key>id: mediumint (not null)
name: varchar(255)
<foreign key>in_adapter_id: tinyint (not null)
<foreign key>out_adapter_id: tinyint (not null)
<foreign key>policy_id: mediumint (not null)
adapters
<primary key>id: tinyint (not null)
name: varchar(30) (not null)
description: varchar(255)
parameters_policies
<primary key>id: mediumint (not null)
<foreign key>parameter_id: mediumint (not null)
<foreign key>policy_id: mediumint (not null)
Value: int (not null)
privatekeys
<primary key>id: smallint (not null)
HSMkey_id: smallint
algorithm: tinyint (not null)
size: smallint (not null)
<foreign key> securitymodule_id: tinyint
generate: timestamp (not null)
<foreign key> policy_id: mediumint (not null)
compromisedflag: tinyint
dnssseckeys
<primary key>id: smallint (not null)
<foreign key>zone_id: smallint (not null)
<foreign key>privatekey_id: smallint (not null)
state: tinyint
publish: timestamp
ready: timestamp
active: timestamp
retire: timestamp
dead: timestamp
keytype: tinyint (not null)
origkeytag: smallint (not null)
revokedkeytag: smallint (not null)
digest1: varchar(20) (not null)
digest256: varchar(32) (not null)
publickey: varchar(??) (not null)
parameters
<primary key>id: mediumint (not null)
name: varchar(30) (not null)
description: varchar(255)
<foreign key>category_id: tinyint
categories
<primary key>id: tinyint (not null)
name: varchar(30) (not null)
securitymodules
<primary key>id: tinyint (not null)
name: varchar(30) (not null)
description: varchar(255)
location: varchar(??) (not null)
capacity: mediumint (not null)
pin: varchar(??)
The id is also referred to by the values of "sm" parameters in the parameters_policies table

9.2 KASP

The KASP data is defined as a set of policies that contain a series of parameters that will control key rollover and the signing of zones.

A policy and the parameters it contains defines the key rollover and signing of a group of zones. You might, for example, have a policy for your large zones and a separate policy for small zones.

All parameters are stored as unsigned 4 byte integers in the range 0 – 4,294,967,295.

Parameters are grouped in to categories according to their use. This is for convenience and to simplify the code. For example, the key life times are stored in a parameter called “lifetime” it may be the lifetime of a KSK or a ZSK according to the category it is in.

Parameter Name	Category	Description	Optional?
refresh	signer	how old a signature may become before it needs to be re-signed	n
jitter	signer	jitter to use in signature inception and expiration times	n
clockskew	signature	estimated max clockskew expected in clients	n
resign	signature	re-signing interval	n
validity	signature	signature validity period	n
version	denial	nsec version (0 or 3)	n
algorithm	denial	nsec3 algorithm	y
iterations	denial	nsec3 iterations	y
optout	denial	nsec3 opt out flag	y
ttl	denial	ttl for nsec3	y
algorithm	ksk	KSK key algorithm	n
bits	zsk	ZSK key size	n
lifetime	ksk	KSK key lifetime	n
security module	ksk	KSK Security Module	n
overlap	ksk	Number of KSK's in use at any one time	n
ttl	ksk	ttl for KSK's	n
rfc5011	ksk	Flag indicating if we are doing RFC5011	n
algorithm	zsk	ZSK key algorithm	n
bits	zsk	ZSK key size	n
lifetime	zsk	ZSK key lifetime	n
security module	zsk	ZSK Security Module	n
ttl	zsk	ttl for ZSK's	n
keycreate	enforcer	policy for key creation 0=fill the hsm, 1=only generate minimum needed	n

The parameters are stored as key value pairs on a per policy basis in the parameters_policies table

(and associated policies, parameters and categories tables). They are stored as key value pairs so that additional parameters can easily be added at a later time.

Two policies will be created at install time:

- A special policy called “opendnssec”
  - Used for parameters that define how the enforcer will run. This is to allow the command line to be minimal.
- A default policy called _default
  - This is to allow users who never need multiple policies to get started easily.

All parameters are required to be present in all policies with the following exceptions.

- The special policy only needs to contain parameters in the enforcer category.
- The NSEC3 parameters may be omitted only if the NSEC version = 0.
- The KSK parameters may be omitted if no KSK's are used.

10 Use cases (sort of)

The Enforcer is a daemon and will run without user intervention. The use cases cover

- How the user configures the Enforcer via the KASP database. It is assumed that a GUI or CLI (referred to just as GUI from now on) will be developed to allow this.
- How automatic events will occur

10.1 Starting the Enforcer

The Enforcer can be run from the command line as follows

ods_enf [OPTION]

Supported options:
-d Debug On.
-h Host.
-u User.
-p Password.
-s Database/Schema Name.
-v Print Version.
-? This Help.

The command line is intended to be minimal. All other configuration MUST be held in the database.

Typically, the enforcer will be started by a init script at system boot time and remain running until system shutdown.

TODO details of init script

10.2 Stopping the Enforcer

The Enforcer MUST respond to a SIGTERM and exit cleanly.

TODO: Describe cleanly – what state must the rest of the system be left in? In particular what state must the signer be in?

10.3 Processing at startup

At startup the Enforcer MUST perform the following in order:

1. parse the command line
2. ensure that it can connect to the database
3. Read any configuration in the special policy called “opendnssec”

If any of these steps encounters an error then the Enforcer will log that error and exit.

10.4 Creation of a policy

A new policy is created by adding a new row to the policies table. The new policy MUST have a name and MAY have a description. When it is created the GUI SHOULD specify default parameters or prompt the user to specify them. All the parameters (as described above) MUST be defined before the policy will be processed. A new policy MAY contain 0 or more zones. A policy will only be used by the Enforcer if

- it contains 1 or more zones
- all the required parameters are present and all are valid

TODO: Define valid

10.5 Setting Parameter Values

Parameters may be set or changed at any time by editing the necessary value in the parameters_policies table. Changing a parameter will take effect next time the policy is scanned by the Enforcer or if the Enforcer daemon is sent a HUP signal.

10.6 Addition of a new Adapter

The adapters indicate to the signer engine how to obtain and serve zone data. To create a new adapter add a row to the adapters table. The following MUST be specified

- name

The following may be specified

- Description

10.7 Addition of a zone to a policy

It is assumed that newly added zones will be unsigned when they are added.

TODO: is that OK?

A zone is added to a policy by creating a row in the zones table and setting the policy_id to the id of an existing policy. The following MUST be set

- name
- in_adapter
- out_adapter
- policy_id

A new zone will start being signed next time the policy is scanned by the Enforcer or if the Enforcer daemon is sent a HUP signal.

10.8 Addition of an HSM to the system

If a new HSM is added to the system it MUST be configured and ready to be used before KASP is told about it.

A new HSM is added to KASP by adding a row to the security modules table. The following MUST be specified:

- name
- location (need to expand on this)
- capacity

The following MAY be specified

- description
- PIN

If the PIN is not specified or if login ever fails then the Enforcer will wait for human intervention. It will prompt for missing PINs

- at every startup
- the first time it encounters a new HSM
- at every failed login.

The Enforcer will NEVER store a PIN that it prompted the user for in the database. The operator MUST use the GUI to add the PIN to KASP if they want login to occur without prompting.

In order for a HSM to actually get used it must be specified in one or more policies using the "security module" parameter as the place for new KSK or ZSKs to be stored.

Whilst the Enforcer is paused waiting for any PIN all key rollover for all policies will stop. Re-signing will continue with the last known keys providing the Signer Engine knows the PIN for the HSM.

10.9 Creating new Parameters

New parameter names may be added to the parameters table. However they will be ignored unless code is written to process them.

10.10 Creating new Categories

New parameter categories may be added to the categories table. However they will be ignored unless code is written to process them.

10.11 Key creation

New keys are created as needed by the Enforcer. However, there is a requirement for new keys to be backed up before they are used. Secure key backup depends on the mechanisms defined by the HSM manufacturer. There is no way to query an HSM for the backup state of a key. Therefore it is assumed that keys are created on a regular basis and that the system administrators will take action to back them up promptly with in a few days of creation. These two intervals

- key generation interval
- backup interval

are defined as parameters for both KSKs and ZSKs in each policy.

TODO: should the backup interval be defined on a per HSM basis?

Keys are created for a particular policy. For example, if (according to the rules below) 10 RSA 1024 bit keys are needed for a policy then 10 RSA 1024 bit keys will be created in the HSM and entries placed in the private keys table. These keys will have the following settings

- id
  - A unique identifier in the table
- HSMkey_id
  - The CKA_ID attribute in PKCS#11
- Algorithm
  - The DNS Algorithm number from RFC4034 Appendix 1. Note, this is the algorithm for the signing but it is assumed and therefore restricted that, for example, that a key marked RSASHA1 will always be used for RSASHA1 signing and not sometimes for RSAMD5.
- Size
  - The size of the key in bits
- securitymodule_id
  - The id of the security module in the security module table where this key can be found
- generate
  - Timestamp showing when this key was created
- policy_id
  - The policy id of the policy this key was created for.

Until the key is actually used and published in a zone(s) no entries are made in the dnsseckeys table.

A key is created in the HSM by calling
- C_GenerateKeyPair

then the SHA1 digest of the private key is calculated by calling
- C_DigestInit
- C_DigestKey
- C_DigestFinal

The CKA_ID attribute of the public and private key objects is then set to the digest using a call to
- C_SetAttributeValue

If either creating the key in the HSM or adding the entry to the privatekeys table fails then that key creation will be rolled back, an error logged and key rollover will stop for all policies. As usual re-signing will continue.

To create a key the Enforcer will check every time it runs (TODO: what is this interval?) to see how long it is since the last time keys were created. If the key generation interval has elapsed then keys will be created and an event logged to inform the administrators.

In addition the Enforcer will check every time that it runs that sufficient keys exist to reach the end of the current key generation interval. If they do not then additional keys will be created and an event logged to inform the administrators.

A key that has existed for less than the backup interval can not be used for signing but could be published in the zone.

TODO: Is that correct?

If key generation fails then an event is logged to inform the administrators and key rollover is paused for all policies that use that HSM. (Re-signing will continue).

10.12 Key Rollover

This includes the initial signing of a zone (Just a special case of rollover).

Key rollover logic is described in draft-morris-dnsop-dnssec-key-timing-00.txt. This logic is implemented in the libksm code. The Enforcer will just periodically ask libksm for the current keys that the zone must be signed with or that must be published in the zone. If rollover is needed the libksm will perform the rollover by updating the dnsseckeys table. In particular it will set the state field and one of the time-stamps (publish, ready,active, retire, dead) to the time that that event happened.

The Enforcer will communicate this information to the Signer Engine so that the zone is re-signed with and contains the appropriate keys.

TODO: The enforcer will run at intervals of ....

10.13 General Running

Once the Enforcer has started and read its configuration then it will process each policy in the order that the database returns them.

For each policy it will:

1. Read all the parameters defined for that policy
2. Check that the parameters are valid and consistent with each other
   - for example that the signature lifetime is > TTL
   - if error then log and ignore this policy
3. Count the number of zones that the policy applies to
   - If 0 then ignore this policy
4. Check that sufficient keys exist to enforce this policy
   4.1 Count the number of unused keys that exist and are allocated to the policy
   4.2 Predict how many should be expected
   4.3 Create any needed
   4.4 Check if it is time to create them anyway
5. Loop over all the zones in this policy

For each zone it will:

TODO

11 Communication with Signer Engine

The Enforcer will write out a configuration data for the signer engine to use. TODO update this section with the latest from the wiki

JOKE: Could even use NETCONF to communicate with the Signer Engine :)

12 Logging

Syslog is to be used for all event logging.

TODO: complete this list

Event	Description
Key gen failed	
Key generation occurred	
Key not available	Keys exist but the backup interval has not elapsed for them
HSM not found	An HSM is defined in a policy but can not be found in the security modules table.
All PKCS11 errors and warning	See pkcs11 documentation
Error parsing command line	
Error connecting to database	
Special policy not found	
special policy invalid	

13 Error Handling

On all errors the Enforcer will pause key rollover and log errors. The Signer Engine will continue re-signing with the last known configuration if according to its rules it is OK to do so.