Lab 10.1 – Implementing DNSSEC

Objective:
Deploy DNSSEC-signed zones.

Background
DNSSEC (or DNS Security Extensions) provide security to the zone files.

Note:
In the steps below, we are using
myzone.net - our domain
db.myzone.net – zonefile for the domain
Kmyzone.net.+005+12345.key/private = ZSK generated
Kmyzone.net.+005+67890.key/private = KSK generated

Steps:

A. **DNSSEC Validation.** To allow your recursive DNS servers to validate DNSSEC-signed zones.

1. Update the DNS configuration. Add options in the configuration file named.conf to allow DNSSEC.

   These options must be enabled:

   ```
   dnssec-enable yes;
   dnssec-validation yes|auto;
   ```

   `dnssec-enable` allows named to respond to DNS requests from DNSSEC-aware clients. The default is yes, but is best added in the named.conf so you know how to turn it off.

   If `dnssec-validation` is set to auto, it defaults to the DNS root zone as the trust anchor.

   If set to yes, a trust anchor must be explicitly configured using the managed-keys or trusted-keys option.

   ```
   managed-keys {
      // root key
      ". Initial-key 257 3 3 "<key-here>"
   };

   trusted-keys {
      // parent zone
      <myzone.net> 257 3 5 "<key-here>";
   };
   ```

2. For the lab, we will use trusted-keys. A file containing the root's public key will be provided. Copy this into your named.conf

B. **Signing the zone**.

1. Generate the key pair.

   This command generates the ZSK:

   ```
   dnssec-keygen –r /dev/urandom –a <algorithm> –b <keysize> \
   –n ZONE <myzone>
   ```

   ex:

   ```
   dnssec-keygen –r /dev/urandom –a rsasha1 –b1024 –n ZONE myzone.net
   ```

   The defaults are RSASHA1 for the algorithm, with 1024 bits for ZSK and 2048 bits for KSK. Since these are all defaults, we can just issue the command:

   dnssec-keygen –r /dev/urandom <myzone>

   The –r option is used as source of randomness since we are working on virtual machines. On a busy server, this may not be necessary.

   This command generates the KSK:

   ```
   dnssec-keygen –a <algorithm> –b <keysize> –f KSK –n ZONE <myzone>
   ```

   Or simply: `dnssec-keygen –f KSK <myzone>`

2. Include the public DNSKEYs in the zone file.

   You can either copy the entire file or reference to it using the $INCLUDE directive. To do the latter, simply add the lines below to the end of your zonefile (db.myzone.net). Note that you are including only the public portion (.key) into the zone file. The private portion (.private) must be kept secure.

   ```
   $INCLUDE "K<myzone>.+005+<id_of_zsk>.key"
   $INCLUDE "K<myzone>.+005+<id_of_ksk>.key"
   ```

3. Sign the zone using the secret keys. The syntax is:

   ```
   dnssec-signzone –o <zonename> –N INCREMENT -f <output-file> -t \
   –k <KSKfile> <zonefile> <ZSKfile>
   ```

ex:

```
dnssec-signzone –o myzone.net –N INCREMENT –f <output-file> -t -k \
Kmyzone.net.+005+12345 db.myzone.net Kmyzone.net.+005+67890
```

In the lab, please don't add the **–f** option. If not specified, the output file will append a `.signed` in the zonefile.

```
<db.myzone.net>.signed
```

Notice that the output file is bigger than the original zone file. Check with the commands `ls –al` or `wc`.

C. Publishing the zone.

1. Reconfigure to load the signed zone. Edit named.conf and point to the signed zone. For example:

```
zone "<myzone>" {
      type master;
      # file "db.myzone.net";
      file "db.myzone.net.signed";
};
```

Change the file to point to the signed zone.

2. Push the DS record up to your parent domain. Open the file dsset-<yourdomain> (ex: dsset-myzone.net). This contains your DS records (see example below).

```
myzone.net.          IN DS 4297 5 1 C5A8C518B2208463F87CB30E35F247DD7EACCDB1
myzone.net.          IN DS 4297 5 2 27E89E4A769F6C6BC889BB6F2E98374CA835D2B8C750D5505F32144E 1E79B881
```

Send this to your parent zone (for the lab, it's the gTLD server). The parent zone will then include the DS record in their zonefile. The $INCLUDE statement can be used at this stage.

```
$INCLUDE "dsset-myzone.net."
```

In the class, securely send the file to the instructors. You may then check if it has been successfully added using dig.

```
dig @nameserver +noadditional DS myzone.net | grep DS
```

3. For slave servers, the configuration is simple.

In the configuration file, add in the options section:

```
dnssec-enable yes;
dnssec-validation auto | yes;
```

Then edit the zone section to point to the new signed zonefile. After reload, verify that this file exists in the folder specified in the config.

```
zone "<myzone>" {
      type slave;
      masters { X.X.X.X; };
       # file "db.myzone.net";
      file "db.myzone.net.signed";
};
```

4. Check if DNSSEC is working using the dig command. The output should show an RRSIG next to the record you asked.

```
dig @localhost +dnssec +multiline myzone.net
dig @localhost +trace +dnssec myzone.net
```

Also check for the AD bit in the message header flags. It should look something like:

```
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 40679
;; flags: qr rd ra ad; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 1
```

D. Signing the reverse zones.

The steps to sign the reverse zones are similar to the instructions in items B and C. Using the IP block assigned to you, create the reverse zones and sign them accordingly.

Sample steps as follows using reverse domain for block 192.168.100.0/24.

1. Generate the keys.

```
dnssec-keygen –r /dev/urandom –a <algorithm> –b <keysize> \
–n ZONE 100.168.192.in-addr.arpa

dnssec-keygen –r /dev/urandom –a <algorithm> –b <keysize> –f KSK –n
ZONE 100.168.192.in-addr.arpa
```

2. Sign the zone

```
dnssec-signzone –o 100.168.192.in-addr.arpa –N INCREMENT –f
<output-file> -t -k \ K100.168.192.in-addr.arpa.+005+12345
db.192.168.100 K100.168.192.in-addr.arpa.+005+67890
```

3. Publish the signed zone

```
zone "100.168.192.in-addr.arpa" {
   type master;
   # file "db.myzone.net";
   file "db.192.168.100.signed";
};
```

4. Verify using dig command.

```
dig @localhost +dnssec +multiline 1.100.168.192.in-addr.arpa

dig -x @localhost 192.168.100.1
```

E. Adding more security with NSEC3.

NSEC records are created to prove the non-existence of a record. It builds a linked list of all the records in the zone file. The problem with this is it allows anyone to list the zone content. This is called "zone walking." Some tools, like the `ldns-walk` (included in the LDNS library), can be used to do exactly this.

NSEC3 can be used to provide more security. It uses a hashing algorithm to output a "hash" to replace the real domain names. This makes it difficult for an attacker, but not totally impossible.

In the steps above, NSEC was used by default. Let us re-do the key generation and signing this time using NSEC3.

1. Generate keys with NSEC3 support.

   To do this, you may use NSEC3RSASHA1 as your algorithm. The easier way to do this is to use -3 option instead. This option allows a few other algorithms such as RSASHA256 and RSAHSHA512 but sets NSEC3RSASHA1 as default.

```
dnssec-keygen -r /dev/random -3 <myzone>

dnssec-keygen -f ksk -r /dev/random -3 <myzone>
```

2. Sign the zone with a salt.

```
dnssec-signzone -A -3 <salt> -o <zonename> -N INCREMENT -f <output-
file> -t -k <KSKfile> <zonefile> <ZSKfile>
```

   The salt is a random hexadecimal number appended to the domain before hashing. It's a public data that is part of the NSEC3PARAM record. It must be changed once in a while or on regular intervals.

   To generate the salt, you can use either of these:

```
date | sha1sum | cut -b 1-16

head -c 1000 /dev/random | sha1sum | cut -b 1-16
```

   Example:

```
dnssec-signzone –A -3 $(head –c 1000 /dev/random | sha1sum | cut –b \
1-16) –o myzone.net –N INCREMENT –f <output-file> -t -k \
Kmyzone.net.+005+12345 db.myzone.net Kmyzone.net.+005+67890
```

To use NSEC3 without a salt, simply use a single dash.

3. Open the signed zonefile (db.myzone.net.signed). Notice that you now have NSEC3 records added with a hash value of the records in the RDATA.

You have reached the end of the exercise.