;login:

THE MAGAZINE OF USENIX & SAGE

November 2000 • volume 25 • number 7

{

THEME ISSUE: SECURITY

edited by Rik Farrow

inside:

SECURING THE DNS

#

USENIX & SAGE

The Advanced Computing Systems Association &
The System Administrators Guild

securing the DNS

by Evi Nemeth

Evi Nemeth is a member of the computer science faculty at the University of Colorado and a part-time researcher at CAIDA, the Cooperative Associationfor Internet Data Analysis at the San Diego Supercomputer Center. She is about to get out of the UNIX and networking worlds and explore the real world on a sailboat.

<evi@cs.colorado.edu>

As our society becomes more and more dependent on the Internet for information, communication, and business, the Domain Name System (DNS), which holds the Internet together, becomes a tasty target for hackers. A hacker who compromises DNS can divert all traffic destined for one host to another host without users ever knowing they have been led astray. The economic impact of such an attack can be huge.

Securing DNS does not stop Web sites from being broken into and defaced, but it does help to guarantee users that they are actually reaching the Web site they asked for. To be totally honest, securing the DNS will guarantee that you reach the correct IP address, but when connecting to this valid IP address, your connection might still be hijacked or mis-routed due to other weaknesses in the infrastructure. To really secure the Internet we need end-to-end authentication and encryption of the data sent over a connection. Securing the DNS via DNSSEC is the first step, as the DNS can then provide a way to distribute the keys required by any end-to-end security mechanisms. DNS is equally important for email, copying files, printing, or any application that uses domain names instead of network addresses.

There are two main points of vulnerability in the DNS system:

- Server-server updates
- Client-server communication

The Internet Software Consortium's BIND implementation addresses these vulnerabilities with separate mechanisms: TSIG/TKEY for server updates and DNSSEC for client-server lookups. The first allows pairs of servers, such as your master server and its slaves, to authenticate each other before exchanging data. TSIG/TKEY uses shared-secret-based cryptography. DNSSEC allows the client not only to authenticate the identity of a server but also to verify the integrity of the data received from that server. It uses public key cryptography. We describe each of these two mechanisms in detail and then look at some of the outstanding issues that are hampering the widespread deployment of secure DNS zones. Some of the material in this article is adapted with permission from Nemeth et al., *Unix System Administration Handbook, Third Edition*, Prentice Hall PTR, (in press). We assume that the reader is somewhat familiar with DNS resource records, the DNS naming hierarchy, `named` (the BIND name-server daemon), and its configuration file /etc/named.conf.

Securing DNS Transactions with TSIG and TKEY

While DNSSEC (covered in the next section) was being specified, the IETF developed a simpler mechanism called TSIG (RFC2845) to allow secure communication among servers through the use of transaction signatures. Access control based on transaction signatures is more secure than access control based on IP source addresses.

Transaction signatures use a symmetric encryption scheme. That is, the encryption key is the same as the decryption key. This single key is called a shared-secret key. You must use a different key for each pair of servers that want to communicate securely. TSIG is much less expensive computationally than public key cryptography, but it is only appropriate for a local network on which the number of pairs of communicating servers is small. It does not scale to the global Internet.

TSIG signatures sign DNS queries and responses to queries, rather than the authoritative DNS data itself, as is done with DNSSEC. TSIG is typically used for zone transfers between servers or for dynamic updates between a DHCP server and a DNS server.

Older versions of BIND do not understand signed messages and complain about them, sometimes to the point of refusing to load the zone.

TSIG signatures are checked at the time a packet is received and are then discarded; they are not cached and do not become part of the DNS data. Although the TSIG specification allows multiple encryption methods, BIND implements only one, the HMAC-MD5 algorithm.

BIND's dnssec-keygen utility generates a key for a pair of servers. For example, to generate a shared-secret key for two servers, serv1 and serv2, use the command

```
# dnssec-keygen -H 128 -h -n serv1-serv2
```

to create a 128-bit key and store it in the file Kserv1-serv2+157+00000.private. The file contains the string "Key:" followed by a base-64 encoding of the actual key.

The generated key is really just a long random number. You could generate the key manually by writing down an ASCII string of the right length and pretending that it's a base-64 encoding of something, or by using mimencode to encode a random string. The way you create the key is not important; it just has to exist on both machines.

Copy the key to both serv1 and serv2 with scp, or cut and paste it. Do not use telnet or ftp to copy the key; even internal networks may not be secure. The key must be included in both machines' named.conf files. Since named.conf is usually world-readable and keys should not be, put the key in a separate file that is included in named.conf. For example, you could put the snippet

```
key serv1-serv2 {
      algorithm hmac-md5 ;
      secret "shared-key-you-generated" ;
} ;
```

in the file serv1-serv2.key. The file should have mode 600 and its owner should be named's UID. In the named.conf file, you'd add the line

```
include "serv1-serv2.key"
```

near the top.

This part of the configuration simply defines the keys. To make them actually be used to sign and verify updates, each server needs to identify the other with a keys clause. For example, you might add the lines

```
server serv2's-IP-address {
      keys { serv1-serv2 ; } ;
} ;
```

to serv1's named.conf file and

```
server serv1's-IP-address {
      keys { serv1-serv2 ; } ;
} ;
```

to serv2's named.conf file. Any allow-query, allow-transfer, and allow-update clauses in the zone statement for the zone should also refer to the key. For example:

```
allow-transfer { key serv1-serv2 ;} ;
```

When you first start using transaction signatures, run named at debug level 1 (-d1) for a while to see any error messages that are generated. Older versions of BIND do not understand signed messages and complain about them, sometimes to the point of refusing to load the zone.

TKEY is an IETF protocol that BIND 9 implements to allow two hosts to generate a shared-secret key automatically without phone calls or secure copies to distribute the key. It uses an algorithm called the Diffie-Hellman key exchange, in which each side makes up a random number, does some math on it, and sends the result to the other side. Each side then mathematically combines its own number with the transmission it received to arrive at the same key. An eavesdropper might overhear the transmission but will be unable to reverse the math.

Securing Zone Data with DNSSEC

DNSSEC is a set of DNS extensions that authenticates the origin of zone data and verifies its integrity by using public key cryptography. That is, the extensions permit DNS clients to ask the questions, "Did this DNS data really come from the zone's owner?" and "Is this really the data sent by that owner?"

DNSSEC provides three distinct services: key distribution by means of KEY resource records stored in the zone files, origin verification for servers and data, and verification of the integrity of zone data. DNSSEC relies upon a cascading chain of trust: The root servers provide validation information for the top-level domains, the top-level domains provide validation information for the second-level domains, and so on.

Public key cryptosystems use two keys: one to encrypt (sign) and a different one to decrypt (verify). Publishers sign their data with a secret "private" key. Anyone can verify the validity of a signature with a matching "public" key that is widely distributed. If a public key correctly decrypts a zone file, then the zone must have been encrypted with the corresponding private key. The trick is to make sure that the public keys you use for verification are authentic. Public key systems allow one entity to sign the public key of another, thus vouching for the legitimacy of the key; hence the term "chain of trust."

Public key systems allow one entity to sign the public key of another, thus vouching for the legitimacy of the key; hence the term "chain of trust."

The data in a DNS zone is too voluminous to be encrypted with public key cryptography – the encryption would be too slow. Instead, since the data is not secret, a secure hash (e.g., an MD5 checksum) is run on the data and the results of the hash are signed (encrypted) by the zone's private key. The results of the hash are like a fingerprint of the data, and the signed fingerprint is called a digital signature.

Digital signatures are usually appended to the data they authenticate. To verify the signature, you decrypt it with the public key of the signer, run the data through the same secure hash algorithm, and compare the computed hash value with the decrypted hash value. If they match, you have authenticated the signer and verified the integrity of the data.

In the DNSSEC system, each zone has its own public and private keys. The private key signs each RRset (that is, each set of records of the same type for the same host). The public key verifies the signatures and is included in the zone's data in the form of a KEY resource record.

Parent zones sign their child zones' public keys. `named` verifies the authenticity of a child zone's KEY record by checking it against the parent zone's signature. To verify the authenticity of the parent zone's key, `named` can check the parent's parent, and so on back to the root. The public key for the root zone would be included in the root hints file.

Signing a Zone

Several steps are required to create and use signed zones. First, you generate a key pair for the zone. For example, in BIND 9,

```
# dnssec-keygen -a DSA -b 768 -n ZONE mydomain.com.
```

or in BIND 8,

```
# dnskeygen -D768 -z -n mydomain.com.
```

The table below shows the meanings of the arguments to these commands.

Argument	Meaning
For dnssec-keygen (BIND 9)	
-a DSA	Uses the DSA algorithm
-b 768	Creates a 768-bit key pair
-n ZONE mydomain.com.	Creates keys for a zone named mydomain.com
For dnskeygen (BIND 8)	
-D768	Uses the DSA algorithm, with a 768-bit key
-z	Creates a zone key
-n mydomain.com.	Creates keys for a zone named mydomain.com

dnssec-keygen and dnskeygen return the following output:

```
algorithm = 003
key identifier = 12345
flags = 16641
```

They also create files containing the public and private keys:

```
Kmydomain.com.+003+12345.key        # public
Kmydomain.com.+003+12345.private    # private key
```

Generating a key with dnssec-keygen can take a long time, especially if your operating system does not have /dev/random to help with generating randomness. It will ask you to type stuff and use parameters from your typing speed and pauses to get the randomness it needs. It might take five minutes, so don't get impatient, and keep typing until it stops echoing dots for each character you type.

The public key is typically $INCLUDEd into the zone file. It can go anywhere after the SOA record, but is usually the next record after SOA. The DNS key record from the .key file looks like this:

```
mydomain.com. IN KEY 256 3 3 BIT5WLkFva53IhvTBIqKrKVgme7...
```

where the actual key is about 420 characters long.

DNSSEC requires a chain of trust, so a zone's public key must be signed by its parent to be verifiably valid. BIND 8 had no mechanism to get a parent zone to sign a child zone's key other than out-of-band cooperation among administrators. BIND 9 provides a program called dnssec-makekeyset to help with this process.

dnssec-makekeyset bundles the keys you want signed (there may be more than just the zone key), a TTL for the resulting key set, and a signature validity period. For example, the command

```
# dnssec-makekeyset -t 3600 -s +0 -e now+864000
Kmydomain.com.+003+12345
```

bundles the public zone key that you just generated with a TTL of 3,600 seconds (one hour) and requests that the parent's signature be valid for ten days starting from now.

dnssec-makekeyset creates a single output file, mydomain.com.keyset. You must then send the file to the parent zone for signing. It contains the public key and signatures generated by the zone keys themselves so that the parent can verify the child's public key. Here is the mydomain.com.keyset file generated by the dnssec-makekeyset command above:

```
$ORIGIN .
$TTL 3600   ; 1 hour
mydomain.com        IN SIG      KEY 3 2 3600 20000917222654 (
                    20000907222654 64075 mydomain.com.
                    BE8V7nicLOARc0PRvBhBMeX7JXL3TdCUBY2Ah313pg+
                    Wq4THOq0U28Q= )
               KEY      256 3 3 (
                    BIT5WLkFva53IhvTBIqKrKVgme7r/tbnBTRkLDDKjGYCnV
                    57TBIeHkZSgbJ7jfYtuTLv4a2OIF5jJDoHD8LEFKNJfboVma
                    8IGmONId2CSfryeuLdLLwW15bhhPHdw+nXWPFB7MY5s
                    bGLkokpuWmyHXkdWThr3A1ICWBs5GQRg8wMaIGOL4d
                    VSUWefQ/g4hGchEq12kieYVE4j9PE5p3uX2BNe0CIGNf05
                    c1VD6kYIn5Ip4hQZGwVL8hpi6NJsxp2U/krtS7GpHN55WA
                    fRY+joQ4AalY3f+AtapkGdV3lHjr1a7LG0qAgFAhNJ2jqKvoB
                    nXbWKKY9AlzMjsyleRdtRqn+V8vY30uTCkaaykrWhtu02QZ
                    pIGuwx294RudyA3gOQgR1aJ+X6BfUmXm2msmmHq//vL
                    mr )
```

In BIND 9, the parent zone uses the dnssec-signkey program to sign the bundled set of keys:

```
# dnssec-signkey mydomain.com.keyset Kcom.+003+56789
```

This command produces a file called mydomain.com.signedkey, which the parent (com) sends back to the child (mydomain.com) to be included in the zone files for mydomain.com. In BIND 8, the parent uses the dnssigner command.

The signedkey file is similar to the keyset file, except that the SIG record is associated with the com zone. Note, in our example, we generated the key for the com zone to use in the dnssec-signkey command; not the real one.

```
$ORIGIN .
$TTL 3600      ; 1 hour
mydomain.com          IN SIG   KEY 3 2 3600 20000917222654 (
                      2000090722265431263com.BAM/WIdPIwY6b4Aj8a5PZ
                      1UHwfo/qKl65HIlpitdvF2UgKaNJVEMSY4= )
                 KEY      256 3 3 (
                      BIT5WLkFva53IhvTBIqKrKVgme7r/tbnBTRkLDDKjGYCn
                      V57TBIeHkZSgbJ7jfYtuTLv4a2OIF5jJDoHD8LEFKNJ
                                              ...
```

Once you have obtained the parent's signature, you are ready to sign the zone's actual data. Add the records from the signedkey file to the zone data before signing the zone. The signing operation takes a normal zone data file as input and adds SIG and NXT records immediately after every set of resource records. The SIG records are the actual signatures, and the NXT records support the signing of negative answers.

Here is a before and after example for our mydomain.com zone:

```
$TTL 3600       ; 1 hour
; start of authority record for fake mydomain.com
@    IN     SOA      mydomain.com. hostmaster.mydomain.com. (
              2000083000       ; Serial Number
              7200             ; Refresh - check every 2 hours for now
              1800             ; Retry   - 30 minutes
              604800           ; Expire  - 1 week (was 2 weeks)
              7200 )           ; Minimum - 2 hours for now

              KEY  256  3  3 (
                   BIT5WLkFva53IhvTBIqKrKVgme7r/tbnBTRkLDDKjGYCnV57TBIe
                   HkZSgbJ7jfYtuTLv4a2OIF5jJDoHD8LEFKNJfboVma8IGmONId2
                   CSfryeuLdLLwW15bhhPHdw+nXWPFB7MY5sbGLkokpuWmyH
                   XkdWThr3A1ICWBs5GQRg8wMaIGOL4dVSUWefQ/g4hGchEq1
                   2kieYVE4j9PE5p3uX2BNe0CIGNf05c1VD6kYIn5Ip4hQZGwVL8h
                   pi6NJsxp2U/krtS7GpHN55WAfRY+joQ4AalY3f+AtapkGdV3lHjr1
                   a7LG0qAgFAhNJ2jqKvoBnXbWKKY9AlzMjsyleRdtRqn+V8vY30u
                   TCkaaykrWhtu02QZpIGuwx294RudyA3gOQgR1aJ+X6BfUmXm
                   2msmmHq//vLmr )

              IN    A        128.138.243.151
              IN    NS       @
              IN    NS       anchor
              IN    NS       zamboni
              IN    MX       10    @
              IN    MX       30    anchor
              IN    LOC      40 00 23.5 N 105 15 49.2 W 1900m

localhost                   IN        A        127.1
anchor                      IN        A        128.138.242.1
                            IN        A        128.138.243.140
                            IN        MX       10  anchor
                            IN        MX       99  @
awesome                     IN        A        128.138.236.20
                            IN        MX       10  awesome
                            IN        MX       99  @
zamboni                     IN        A        128.138.199.7
                            IN        A        128.138.243.138
                            IN        MX       10  zamboni
                            IN        MX       99  @
```

In BIND 8, you use the dnssigner program in the contrib directory of the distribution to sign a zone; in BIND9, you use the dnssec-signzone command. For example, the command

```
# dnssec-signzone mydomain.com Kmydomain.com.+003+12345 # BIND 9
```

reads the zone file mydomain.com and produces a signed version using the private key in the Kmydomain.com+003+12345.private file. The resulting file is called mydomain.com.signed. If you forget to include the key file on the command line, you get an obscure error message about an "inappropriate ioctl for device" from the module entropy.c.

It can take a long time to sign a zone, especially if your system does not have /dev/random, because it asks you to type a few sentences for each signature it generates. Quite a pain after a while. If you try to fool it by cutting and pasting text in, it makes you type more till it feels there has been sufficient randomness generated. Here is a portion of the resulting signed zone file for mydomain.com:

```
$ORIGIN .
$TTL 3600      ; 1 hour
mydomain.com           IN SOA  mydomain.com. hostmaster.mydomain.com. (
                       2000083000  ; serial
                       7200        ; refresh (2 hours)
                       1800        ; retry (30 minutes)
                       604800      ; expire (1 week)
                       7200        ; minimum (2 hours)
                       )
               SIG        SOA 3 2 3600 20001008023531 (
                       2000090802353164075mydomain.com.BFN/8mIRX/M
                       W01kMoe+7qId63LB7Tbb9t98/NnfY16WQgItk03FDXTk
                       = )
               NS         mydomain.com.
               NS         anchor.mydomain.com.
               NS         zamboni.mydomain.com.
               SIG        NS 3 2 3600 20001008023531 (
                       20000908023531 64075 mydomain.com.
                       BAkrse9uTdANxbGA0kaWkjiippeCCUBcvHGR7zDOt+k
                       STeGbVfJy8iw= )
               SIG        LOC 3 2 3600 20001008023531 (
                       20000908023531 64075 mydomain.com.
                       BIARXt5zqiPy08Ca7T7AiUCau1PJEWlv3uHTQci0f3g5nlr
                       kw1exaqM= )
               SIG        MX 3 2 3600 20001008023531 (
                       2000090802353164075mydomain.com.
                       BEHMocIH/p1cL0FlQTz1cfEZzqHHHf1BBLSy2FtU1H6v
                       5DXZy9zkyOw= )
               SIG        A 3 2 3600 20001008023531 (
                       20000908023531 64075 mydomain.com.
                       BGKrpBrAkCtHcuzX57heH5sS0MYnFRC3MqeRMf3i881
                       Y3ZD+Q+E9r24= )
               SIG        KEY 3 2 3600 20000917222654 (
                       20000907222654 31263 com.
                       BAM/WIdPIwY6b4Aj8a5PZ1UHwfo/qKl65HIlpitdvF2UgK
                       aNJVEMSY4= )
$TTL 7200      ; 2 hours
               SIG        NXT 3 2 7200 20001008023531 (
                       20000908023531 64075 mydomain.com.
                       BDvw+QgYBcmIXeS4qMyMNDtB8K+sX5Jb2zKMRCcQ
                       4uFySJJVQ/s6A1w= )
               NXT        anchor.mydomain.com. ( A NS SOA MX SIG
                          KEY LOC N XT )
$TTL 3600      ; 1 hour
               KEY        256 3 3 (
                       BIT5WLkFva53IhvTBIqKrKVgme7r/tbnBTRkLDDKjGYCn
                       V57TBIeHkZSgbJ7jfYtuTLv4a2OIF5jJDoHD8LEFKNJfbo
                       Vma8IGmONId2CSfryeuLdLLwW15bhhPHdw+nXWPFB
                       7MY5sbGLkokpuWmyHXkdWThr3A1ICWBs5GQRg8w
                       MaIGOL4dVSUWefQ/g4hGchEq12kieYVE4j9PE5p3uX2
                       BNe0CIGNf05c1VD6kYIn5Ip4hQZGwVL8hpi6NJsxp2U/
                       krtS7GpHN55WAfRY+joQ4AalY3f+AtapkGdV3lHjr1a7L
                       G0qAgFAhNJ2jqKvoBnXbWKKY9AlzMjsyleRdtRqn+V8v
                       Y30uTCkaaykrWhtu02QZpIGuwx294RudyA3gOQgR1aJ
                       +X6BfUmXm2msmmHq//vLmr )
```

```
                A           128.138.243.151
                MX          10 mydomain.com.
                MX          30 anchor.mydomain.com.
                LOC         40 0 23.500 N 105 15 49.200 W 1900.00m 1m
                            10000m 10m
$ORIGIN mydomain.com.
anchor          SIG         MX 3 3 3600 20001008023531 (
                         20000908023531 64075 mydomain.com.
                         BFEtOCT+y0dQPx7Am7gpxD9SjEl+USuaE7qExUOrX22
                         X7wjqJFJbqdo= )
                SIG         A 3 3 3600 20001008023531 (
                         20000908023531 64075 mydomain.com.
                         BDwfBm2j6xFLoXttzvtuln9ZD+9qUWBAwSBJVB06WJ/
                         Rc6+F1ubj/fs= )
$TTL 7200       ; 2 hours
                SIG         NXT 3 3 7200 20001008023531 (
                         20000908023531 64075 mydomain.com.
                         BIMwxryI8NyfWupBe4JJmeRCCj1/FnyPjxAuBOQKTRX
                         X4FsaDrma1X4= )
                NXT         awesome ( A MX SIG NXT )
$TTL 3600       ; 1 hour
                A           128.138.242.1
                A           128.138.243.140
                MX          10 anchor
                MX          99 mydomain.com.
                                        ...
```

The signedkey file from the parent domain .com gets slurped into the processing if it is in the same directory as the zone file being signed (see the SIG KEY record associated with com toward the top of the example). There is quite an increase in the size of the zone file, roughly a factor of three in our example. The records are also reordered.

A SIG record contains a wealth of information:

- The type of record set being signed
- The signature algorithm used (in our case, it's 3, the DSA algorithm)
- The TTL of the record set that was signed
- The time the signature expires (as yyyymmddhhssss)
- The time the record set was signed (also yyyymmddhhssss)
- The key identifier (in our case 12345)
- The signer's name (mydomain.com.)
- And finally, the digital signature itself

To use the signed zone, change the file parameter in the named.conf zone statement for mydomain.com to point at mydomain.com.signed instead of mydomain.com. In BIND 8, you must also include a pubkey statement in the zone statement; BIND 8 verifies the zone data as it loads and so must know the key beforehand. BIND 9 does not perform this verification. It gets the public key from the KEY record in the zone data and does not need any other configuration.

Whew! That's it.

Negative Answers

Digital signatures are fine for positive answers like "Here is the IP address for the host anchor.mydomain.com, along with a signature to prove that it really came from mydomain.com and that the data is valid." But what about negative answers like "No such host"? Such negative responses typically do not return any signable records.

In DNSSEC, this problem is handled by NXT records that list the next record in the zone in a canonical sorted order. If the next record after anchor in mydomain.com is awesome.mydomain.com and a query for anthill.mydomain.com arrived, the response would be a signed NXT record such as

```
anchor.mydomain.com. IN  NXT  awesome.mydomain.com ( A MX SIG NXT )
```

This record says that the name immediately after anchor in the mydomain.com zone is awesome, and that anchor has at least one A record, MX record, SIG record, and NXT record. The last NXT record in a zone wraps around to the first host in the zone. For example, the NXT record for zamboni.mydomain.com points back to the first record, that of mydomain.com itself:

```
zamboni.mydomain.com. IN  NXT  mydomain.com. ( A MX SIG NXT )
```

NXT records are also returned if the host exists but the record type queried for does not exist. For example, if the query was for a LOC record for anchor, anchor's same NXT record would be returned and would show only A, MX, SIG, and NXT records.

We have described DNSSEC as of BIND v9.0.0rc5 (September 2000). Judging from the significant changes that occurred during the beta cycle, this information may not be current for long. As always, consult the documentation that comes with BIND for the exact details.

Outstanding Issues

Now that we have described the two mechanisms available in BIND for securing the DNS, let's look at some of the potential problems.

CACHING AND FORWARDING

DNSSEC is at odds with the notions of caching and forwarders.

DNSSEC is at odds with the notions of caching and forwarders. DNSSEC assumes that queries contact the root zone first and then follow referrals down the domain chain to get an answer. Each signed zone signs its children's keys, and the chain of trust is unbroken and verifiable. When you use a forwarder, however, the initial query is diverted from the root zone and sent to your forwarding server for processing. A caching server that is querying through a forwarder will recheck signatures, so responses are guaranteed to be secure. But, for the query to succeed, the forwarder must be capable of returning all the SIGs and KEYs needed for the signature checking. Non-DNSSEC servers don't know to do this, and the RFCs ignore the whole issue of forwarding.

BIND 9 implements some extra features beyond those required by RFC2535 so that a BIND 9 caching server can use DNSSEC through a BIND 9 forwarder. If you are using forwarders and want to use DNSSEC, you might have to run BIND 9 throughout your site.

Unfortunately, those busy sites that use forwarders and caching are probably the sites most interested in DNSSEC. Alas, the standards writers didn't quite think through all of the implications for the other parts of the DNS system.

Public Key Infrastructure

DNSSEC also relies on the existence of a public key infrastructure that isn't quite a reality yet. There is no smooth way to get the parent to sign a child's keys; we cannot yet send mail to the hostmaster@com and get signed keys back. A public key infrastructure

NLnet Labs is currently running an experiment on DNSSEC issues in a special domain called NL.NL. They are building tools to automate the signing of subdomain keys and helping people get their domains secured.

is needed for other applications too, such as IPSec (a secure version of the IP protocol) or e-commerce. DNSSEC is the first step in the chain of a series of security enhancements being deployed on the Internet.

The private key for the root zone is essential for the whole process to work. It must be used whenever the root zone changes and needs to be re-signed. Therefore it must be accessible. What do we do if the private root key is lost or stolen? Generating a new key pair is fast, but distributing it to millions of DNS servers isn't. How do we change keys? There must be some period of time during which both the old key and the new one are valid if caching is to work. How do we plan for changing important keys – the root, the key for the COM zone, etc.? How do we engineer the switchover to not destabilize the network? If the root key is built into the software, then a compromise implies that every DNS server out there must be manually touched and changed. The disruption to the network would be worse than the damage that whoever stole the root key could do.

SIGNING BIG ZONES

The COM zone is over 2GB. It is typically updated twice a day. But with current hardware and software-signing the COM zone takes several days. An incremental mechanism for re-signing a zone is built into the current BINDv9 distribution. Re-signing, when not much has changed, takes about 5% of the time to do the original signing. We are within striking distance of being able to maintain a signed copy of the COM zone.

The folks at NLnet Labs (<*http://www.nlnetlabs.nl/dnssec*>) have experimented with signing the top-level DE, NL, ORG, and COM zones. Some of their results are shown below:

DE	full zone	13 hours	FreeBSD 3.4 PC
DE	delegation zone	4 hours	FreeBSD 3.4 PC
ORG	full zone	42 hours*	Red Hat Linux 6.2, DEC/Compaq Alpha
COM	delegation zone	50 hours	Red Hat Linux 6.2, DEC/Compaq Alpha

* It took 2 hours to re-sign after 1 record changed

The COM zone snapshot included 12 million delegations and was sorted before signing (three hours with the standard UNIX `sort` command). The process used 9GB of virtual memory, and it took an additional nine hours to write the signed zone out to disk.

NLnet Labs is currently running an experiment on DNSSEC issues in a special domain called NL.NL. They are building tools to automate the signing of subdomain keys and helping people get their domains secured.

PERFORMANCE

Signed zones are bigger. Signed answers to queries are bigger. UDP, the default transport protocol used by DNS, has a limitation that makes the maximum packet size 512 bytes in the default case. If an answer is greater than 512 bytes, a truncated answer comes back in a UDP packet and the client re-asks the query using TCP. TCP is slower and requires more network traffic. It's unclear whether the current servers for COM could keep up with the query rate if a large portion of the DNS traffic were TCP.

Verifying signatures also costs CPU time and memory; the cost is about one-twentieth of the cost of signing. The actual rates for both signing and verifying depend on the encryption algorithm used, with DSA-512 the fastest (signing about 135 domains/sec. on a 500Mhz FreeBSD PC) and RSA-1024 the slowest (17 domains/sec.). DSA-768, a popular algorithm, is in the middle at 62 domains/sec.

Transaction signatures (TSIG/TKEY) use less CPU time and network bandwidth than do public key authentication methods, but they guarantee only that you know where your responses came from, not that the responses are correct. A combination of a TSIG relationship with a server known to do full DNSSEC might provide a reasonable degree of security. It is not possible to have a TSIG relationship with every server you might ever want to talk to, since TSIG relationships must be manually configured.

Look for DNSSEC to be fully deployable in the near future.

Conclusions

The ISC BIND version 9 contains an initial set of tools for sites to begin securing their DNS. However, the public key infrastructure and automated mechanisms for a child zone to have its key signed by its parent are not yet in place. Look for DNSSEC to be fully deployable in the near future – it is the key ingredient in a public key infrastructure that can be used by any application requiring authentication, security, or privacy. Folks with very sensitive data (banks, e-commerce sites, military installations, etc.) might want to start experimenting with DNSSEC now, at least within the corporate intranet.