DNS Security

Antonio Lioy, Fabio Maino, Marius Marian, Daniele Mazzocchi
Dipartimento di Automatica e Informatica
Politecnico di Torino
Torino (Italy)

Abstract— **The present article is an overview of the security problems affecting the Domain Name System and also of the solutions developed throughout the last years in order to provide a better, trustworthy and safer name resolution protocol for the expanding Internet community of present days.**

The paper discusses the basic notions regarding DNS and introduces the reader to the known security threats regarding DNS. The DNSSEC subset proposed is presented and analyzed from both theoretical and practical points of view, explaining the existing security features of the implementations available today.

Keywords— **name resolution, name server, DNS security, public key infrastructure.**

I. Introduction

To fully understand the strategic relevance of the DNS security it worths reminding a well-known case of DNS spoofing. In June 1997, Alternic (an alternative name registry) redirected the *internic.net* domain (Internic - the main name registry of names in Internet) to their site, *www.alternic.net*. All started from the fact that many Internet users feel that the Internic control of the top level domains was against the spirit of Internet. Alternic stated that they were protesting the Internic's claim to ownership of *.com*, *.org* and *.net* top level domains which they were supposed to be running on public trust. As a consequence, in October 1997, Eugene Kashpureff - Alternic founder - was arrested by Canadian authorities and faced with extradition to the U.S. in conjunction with wire fraud. He was the brain behind the attack that programmed the Internic name server to route all viewers to the Alternic site. Kashpureff succeeded to cache bogus information on the target name server rerouting *www.internic.net* to *www.alternic.net* and *www.netsol.com* to *www.alternic.net*, as well as causing failures to resolve other addresses.

Since that time, no essential changes were made for the DNS security so that this type of attacks are still possible today. Furthermore, the Domain Name System is used extensively by almost all the applications and protocols that are involved in network communication. Therefore it was predictable that at some point someone will observe the weaknesses of DNS and will take advantage of them. At present, DNS spoofing, DNS cache poisoning are still happening and are getting quite annoying for system administrators responsible for the domains being spoofed. In this paper, we will present the features that will secure the Domain Name System and why these security extensions are necessary. Section 2 introduces the Domain Name System fundamentals. The brief overview is meant to familiarize the reader with the basic notions used in the article. Section 3 describes and classifies the weaknesses of the DNS: misdirected destination, misdirected source and other DNS based attacks are described. Section 4 deals with the security solutions globally referred as DNSSEC, and defined in the set of documents that includes Request For Comments 2535 through 2537. A special consideration is given to the TSIG (Transaction SIGnature) resource record regarded as a complementary security enhancement of DNS. A relevant aspect of the DNSSEC architecture is its capability of storing public keys, hence acting as a public key infrastructure. The last section is reserved for a critical analysis of DNSSEC.

II. What is DNS?

DNS is the shorthand for the Domain Name System. The Domain Name System provides a mechanism of conversion with a double functionality: it translates both symbolic host names to IP addresses and IP addresses to host names.

The DNS has three major components:

- The first category contains:
  - the **Domain Name Space** and
  - the **Resource Records**, that are specifications for a tree structured name space and the data associated with these names.
- **Name Servers** are server programs which maintain the information about the DNS tree structure and can set information. A name server may cache information about any part of the domain tree, but in general it has complete information about a specific part of the DNS. This means the name server has authority for that subdomain of the name space - therefore it will be called *authoritative*.
- **Resolvers** are programs that extract the information from name servers in response to client requests.

It is assumed that the reader is familiar with the basic notions about DNS. A more detailed presentation of the DNS can be found in the appendix.

III. DNS Security Problems

It is known the fact that DNS is weak in several places. Using the Domain Name System we face the problem of trusting the information that came from a non authenticated authority, the name-based authentication process, and the problem of accepting additional information that was not requested and that may be incorrect.
"Many of the classic security breaches in the history of computers and computer networking have had to do not with fundamental algorithm or protocol flaws, but with implementation errors. While we do not intend to demean the efforts of those involved in upgrading the Internet protocols to make security a more realistic goal, we have observed that if BIND would just do what the DNS specifications say it should do, stop crashing, and start checking its inputs, then most of the existing security holes in DNS *as practiced* would go away." - Paul Vixie, founder of ISC and main programmer of BIND.

A. Misdirected Destination: Trusting Faked Information

Suppose the following scenario: a user wants to connect to host A by means of a telnet client. The telnet client asks through a resolver the local name server to resolve the name A into an IP address, it receives a faked answer, and then initiates a TCP connection to the telnet server on the machine A (so it thinks). The user sends his login and password to the fake address. Now, the connection drops and the user retries the whole procedure this time to the correct IP address of the host A. He might ignore what just happened but the malicious attacker that spoofed the name of the host A is now in control of his login and password. This happened because the present routers have no capacity to disallow packets with fake source addresses. So, if the attacker can route packets to someone, then he is capable of forging those packets to look as if they come from a trustworthy host. Therefore, in our case the attacker predicts the time when a query will be sent and he starts to flood the resolver with his fake answers. With a firewall for the user's network the resolver would not be reachable from the outside world, but his local name server would. So, if the local name server can be corrupted in the same manner as described above then the attacker can redirect such application with vital information towards hosts controlled by him and capture these information. Following these assumptions, we observe that in this case we have the possibility of a Denial of Service (DoS) attack. In case of such an attack, if the name server can be spoofed and the attacker's machine can impersonate the true name server then it can maliciously provide that certain names in the domain do not exist. Later on, we present a way in which such an attack is annihilated in DNSSEC.

B. Name Based Authentication/Authorization

Some applications, unfortunately spreaded all over the Internet, make use of an extremely insecure mechanism: name based authentication/authorization. It is the case, for example, of the Unix "r-commands" such as `rlogin`, `rsh` or `rcp` that use the concept of "remote equivalence" to allow the remote access to a computer.

In these networks, system administrators or, even worse, users can declare the remote equivalence of two accounts on two different machines (e.g., by means of the files */etc/hosts.equiv* or *.rhosts*). This equivalence associates two users of two different hosts simply on the basis of their names. The access to a remote computer is then granted if the remote user is declared equivalent to a local user, and if the requesting hostname matches the one contained in the equivalence definition. No other authentication mechanisms are used, so we can talk of name based (weak) authentication. As an example, user *joe* can login as the user *doe* to the computer *host.mydomain.com* from the computer *otherhost.mydomain.com* if the file */etc/hosts.equiv* contains the equivalence between the local user *doe* and the user *joe@otherhost.mydomain.com*.

Remote commands have been designed at the dawn of the Internet for the use in trusted local network, where all the users were known to the system administrator, and the network was not connected to the big Internet. Unfortunately, remote commands survived to the Internet growth and they are still present and used in many networks.

If name based authentication/authorization is used, it is possible to access to a remote machine simply spoofing the name of a host. Also, if the local network is protected by a firewall, all the hosts that use name based authentication/authorization are at risk if an attacker can get control of a single machine of the firewall-protected network. The attacker can monitor network traffic learning the equivalences used in that network, and spoof the IP address of an equivalent host (e.g., performing a denial of service attack on that machine, or simply waiting for the machine to shut-down). Now, the attacker's host is completely equivalent to the spoofed host for all the computers using remote equivalence.

C. Trusting Supplementary Non-Authoritative Information

This is another side of the DNS weakness. For the goal of efficiency the DNS was designed to have the additional

section in its standard message format. Therefore, in certain cases when a supplementary information is considered necessary to speed up the response for a given query, it is included in the additional section. One example, if we ask our name server, i.e. *ns.mydomain.com*, to retrieve the mail exchange RR for the domain *comp-craiova.ro*, the server responds with a *mail.gate.comp-craiova.ro* RR in the answer section of the DNS message and in the additional section the name and the IP address of the name server authoritative for the *comp-craiova.ro* domain are included.

Remember that this information was not explicitely asked by us, rather it was cached by our name server, in its pursuit for solving our query, in order to avoid further lookups for the name server authoritative for that domain.

The type of attack possible in this case is called "cache poisoning". How does this happen? Suppose the following situation described in figure 1. An attacker controlling the name server for his domain *evil.com* wants to poison the cache of another name server called *ns.broker.com* used by a broker's agency in order to impersonate the machine *www.bank.com* that is often accessed by the users in the domain *broker.com*. From his machine the attacker asks the name server *ns.broker.com* for a name under the authority of his own name server, e.g. *anyhost.evil.com*. The name server *ns.broker.com* will contact the attacker's name server - authoritative for that name. This name server will answer the query and will also get the query ID which it stores for later use. This query ID is placed in the header section of any DNS message and is assigned by the program that generated the query (i.e., the target name server). This identifier is copied in the corresponding reply and can be used by the requester to match up replies to outstanding queries - as mentioned in the RFC 1035 [2]. The attack continues with another query from the attacker's side. He knows that the broker's agency is frequently contacting a certain bank site whose name he is willing to spoof. Therefore, he will ask the *ns.broker.com* name server for the address of the *www.bank.com*. Normally, the name server *ns.broker.com* will contact the DNS server authoritative for the domain *bank.com* (e.g., *ns.bank.com*). At this point, the attacker will start to flood the *ns.broker.com* server with replies in which the address of the attacker's machine is mapped to the name *www.bank.com* before the true response can arrive from the authoritative name server (that is *ns.bank.com*). He also can predict correctly the query and reply ID, since he already has the last query ID generated by *ns.broker.com*. In this way, the server *ns.broker.com* receives an information which is not proper and also caches it after responding to the former attacker's query for *www.bank.com*. Now, the trap is set and all the attacker has to do is wait until a connection from *broker.com* domain is made to *www.bank.com*. Since the IP address of the attacker's machine is mapped incorrectly, in the server's cache (*ns.broker.com*), to the name *www.bank.com* all the connections to the bank will be directed to the attacker's machine. The name server will not try to query again for *www.bank.com*, it will just use the information it cached during previous DNS lookups. This is another reason why data received by the name servers in the DNS need origin authentication and integrity verification.

IV. DNS Security Extensions

In the RFC 1035 [2] - "DNS implementation and specification" - the security considerations were not forgotten since it is emphasized that the cache integrity is of maximum importance. Despite this statement, the need for performance has pushed the present implementations to the situation of adding unauthorized records to the additional section and - lacking a strong authentication mechanism - believing that all information provided by DNS is trustworthy.

A. Involving Cryptography

With RFC 2065 [3] and afterwards with its successor RFC 2535 [4], the need for security extensions to DNS was acknowledged and standardized in an organized manner within the DNSSEC IETF working group. The first step is to provide data authentication of the resource records travelling back and forth in the internet. With authentication come also data integrity and data source authentication. The authentication is obtained by means of cryptographic digital signatures. The public key algorithms used for authentication in DNSSEC are MD5/RSA and DSA. The digital signatures generated with public key algorithms have the advantage that anyone having the public key can verify them. Each resource record in the DNS messages exchanged can be digitally signed providing data origin authentication and integrity of the message.

In addition, DNSSEC defines new resource records for storage of public keys in the DNS. These RRs can be used to distribute the keys involved in the security of the DNS itself, but also to distribute keys associated with names to support other security aware protocols (e.g., IPSEC). In the following, we will examine the proposed extensions. First of all, the resource records added for authentication support are KEY and SIG. The KEY RR contains the public key for a host or for a zone. The SIG RR contains the digital signature associated with each set of records.

For a signed zone, there is a zone KEY RR and each resource record in the zone is signed with the zone's corre-

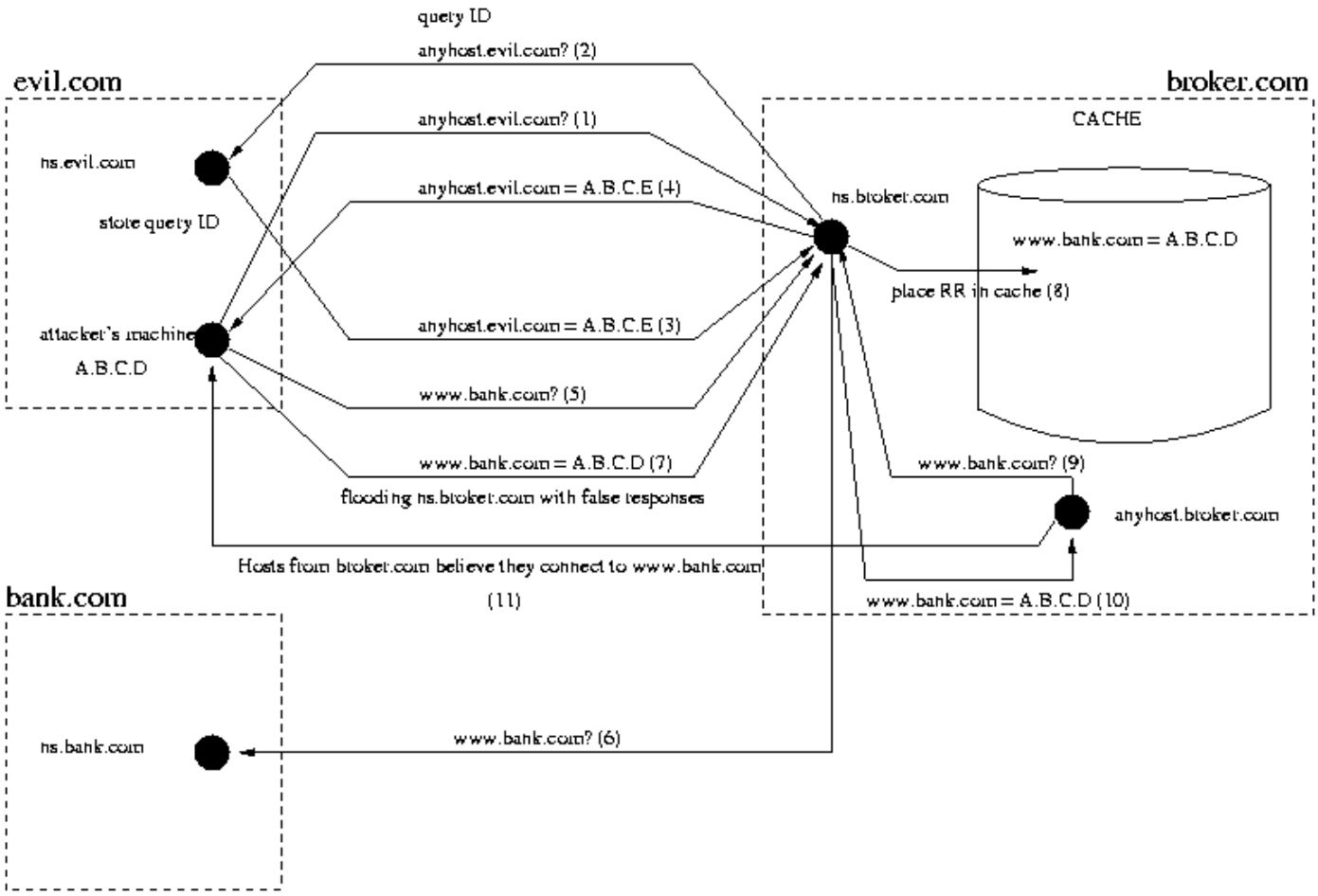

Figure 1. A cache poisoning example

sponding private key. To better understand this statement, we figure below the differences in a zone file between standard DNS and secure DNS. For standard DNS, we have the following resource records in the sample zone data file as seen in figure 2. We have a simple zone data file in which are present the resource records specifying the start of authority (SOA) parameters, the name server (NS), the mail exchange (MX) for the domain *mydomain.com* (lines 1-9 in figure 2). We also have the IP addresses for the name server, mail exchange box and one host in *mydomain.com* - given by the A RR.

When DNSSEC is used, the zone file modifies as presented in figure 3. First of all, we can see that for each resource record from figure 2 we have an acompanying SIG RR in figure 3. Each resource record in the old zone file is signed by the zone's private key. The SIG RR cryptographically binds a resource record set to the signer and a validity interval. In our example, the validity interval of the signature is marked by the values 19991023133034 (the time of expiring: 13:30:34 on the 23rd of October 1999) and 19990923133046 (the time of signing the zone: 13:30:46 on the 23rd of September 1999). Also, notice the different SIG RR that appears in line 13 of figure 3. It refers to AXFR RR. This record was not present in the figure 2. This SIG is meant to efficiently assure the completeness and security of zone transfers. The AXFR signature must be calculated after all the RRs in the zone file are signed and inserted. In fact, it belongs to the zone as a whole, not only to the zone name. This AXFR signature is retrieved only as part of a zone transfer. The signatures of the RRs and of the zone data file are generated by a special program: the zone signer. The task of doing the signing is charged to the zone administrator. The zone signer reads all the data in the zone file, organizes it in a canonical order, arranges related records into groups, signs them all and adds the SIG and NXT RRs at their proper places. At the end all this information is written in a file that will be later on used by the zone primary name server.

The use of the NXT RR is meant to authenticate the non-existance of a resource record. As it can be seen the structure of the file by using the NXT RR becomes circular in the way that if someone looks at the last NXT RR in the zone, he will see that it points to the RRs of name *mydomain.com.* which is in the beginning of the file. It

Line	Zone Data File				
1:	mydomain.com.	IN	SOA	securens.mydomain.com.	sysadmin.mydomain.com.(
2:				19991011	;serial number
3:				86400	;refresh time
4:				1800	;retry time
5:				2592000	;expire time
6:				86400	;time to live
7:				)	
8:	mydomain.com.	IN	NS	securens.mydomain.com.	
9:	mydomain.com.	IN	MX	10 servmail.mydomain.com.	
10:	host	IN	A	131.87.24.3	
11:	securens	IN	A	131.87.24.1	
12:	servmail	IN	A	131.87.24.4	

Figure 2. A zone file in the classic DNS

is also important to observe that the names in the zone are arranged in canonical order (with one exception: the zone name). The NXT RR plays the role of linking successive names in a zone file. As seen in figure 3, for each name in the zone file we have one NXT RR pointing to the next name in the zone and also specifying the type of RRs. For example, the NXT RR for *mydomain.com* points to *host.mydomain.com* (next name in the zone) and specifies as existing types of RRs for *mydomain.com* the NS, SOA, MX, SIG and NXT. This is necessary to provide a mechanism by which someone could verify the non-existance of a name in a zone or the non-existance of a type for an existing name. If a resolver asks for a name that does not exist in the zone file, the DNSSEC server authoritative for that domain will return a signed SOA RR and also a NXT RR that will authenticate the non-existance of that name. An example, if a resolver asks for the name *newhost.mydomain.com* the A (address) RR, the authoritative name server for the domain *mydomain.com*, i.e., *securens.mydomain.com*, will return a signed SOA RR together with the NXT RR of the *host.mydomain.com* (i.e., host IN NXT securens.mydomain.com A SIG NXT).

What does it mean? The interpretation is that between *host.mydomain.com* and *securens.mydomain.com* there are no other names in the zone file and only the A, SIG and NXT RRs are available for that name (i.e., for *host.mydomain.com*). Therefore, the resolver by verifying the signature of the NXT RR can easily infer that the name it requested does not exist and only these three types of RRs are available for that name. The NXT RR provides for defense against replay attacks. Moreover, the NXT RR will be included in the authority section of responses and this is a change in the existing standard (RFC 1034/1035 [1] [2]) which included only SOA and NS RRs in the authority section. In short, the NXT RR is used to securely indicate that RRs with an owner name in a certain name interval do not exist in a zone and to indicate which RRs are associated to an existing name in the zone. The NXT RR must be automatically computed and added during the signing of the zone and they must be signed by the zone's key.

B. Chaining through KEYs

When a resolver receives a response from a DNSSEC name server, it must start the verification of the signatures associated to the RRs received as answers. But the verification of the signature only says that the message was correctly signed. It doesn't say anything about trusting or not the data. So, we only have data integrity, but not data origin authentication. The resolver must somehow determine if the KEY that signed the RRs is trustworthy and if it is authorized to sign those RRs. In order to do this it must build a cryptographically verified chain of KEY and SIG RRs to a point of trust. Each KEY RR is signed by the parent zone's KEY. So, in order to verify the SIG RR of a KEY the resolver must retrieve additional information about parent zones.

Let's see an example. Suppose you want to retrieve the address RR of *host.mydomain.com*. Your resolver would query your security aware name server for the name and type requested. When receiving the response, you have an A RR for the name *host.mydomain.com* and also a SIG RR for the A RR together with a KEY RR containing the public key that verifies the signature. The problem raised is can you trust that public key?

The process of authentication is based on the following facts. The root public key is trusted since it is preconfigured in the resolver. Let's assume that the resource records of *mydomain.com* (placed on the machine *securens.mydomain.com*) were signed with the private key of the *mydomain.com*. The public key (stored in the KEY RR retrieved by you) is also signed, but by the parent do-

Line	Zone Data File				
1:	mydomain.com.	IN	SOA	securens.mydomain.com.	sysadmin.mydomain.com. (
2:				938093434	;serial number
3:				86400	;refresh time
4:				1800	;retry time
5:				2592000	;expire time
6:				86400	;TTL
7:				)	;Cl=2
8:	mydomain.com.	IN	SIG	SOA 1 86400	(;RR type, alg. type, TTL
9:				19991023133034	;SIG expiration time
10:				19990923133041	;SIG inception time
11:				59104 mydomain.com.	;key tag, signer's name
12:				aMA12laNfGhi/ukl...	);the signature
13:	mydomain.com.	IN	SIG	AXFR 1 86400	(;zone transfer signature
14:				19991023133034 19990923133046 59104 mydomain.com.	
15:				rEma121ANfasdc...	);the signature
16:	mydomain.com.	IN	NS	securens.mydomain.com.	
17:	mydomain.com.	IN	SIG	NS 1 86400	(
18:				19991023133034 19990923133041 59104 mydomain.com.	
19:				BmmAR12LenCDFsS...	);the signature
20:	mydomain.com.	IN	MX	10 servmail.mydomain.com.	
21:	mydomain.com.	IN	SIG	MX 1 86400	(
22:				19991023133034 19990923133041 59104 mydomain.com.	
23:				kjldfieVDSl...	)
24:	mydomain.com.	IN	NXT	host.mydomain.com.	NS SOA MX SIG NXT
25:	mydomain.com.	IN	SIG	NXT 1 86400	(
26:				19991023133034 19990923133041 59104 mydomain.com.	
27:				LfDS5Almcds21...	)
28:	host	IN	A	131.87.24.3	
29:		IN	SIG	A 1 86400	(
30:				19991023133034 19990923133044 59104 mydomain.com.	
31:		IN	NXT	securens.mydomain.com.	A SIG NXT
32:		IN	SIG	NXT 1 86400	
33:				19991023133034 19990923133044 59104 mydomain.com.	
34:				LdSD+/34mCDGfy...	)
35:	securens	IN	A	131.87.24.1	
36:		IN	SIG	A 1 86400	(
37:				19991023133034 19990923133046 59104 mydomain.com.	
38:				HgfT4K08VBDliv...	)
39:		IN	KEY	KmOP/sd7REvb3Kii...	;the securens public key
40:		IN	SIG	KEY 1 86400	(
41:				19991023133034 19990923133046 59104 mydomain.com.	
42:				OrT2M09/xZE...	)
43:		IN	NXT	servmail.mydomain.com.	A SIG KEY NXT
44:		IN	SIG	NXT 1 86400	
45:				19991023133034 19990923133046 59104 mydomain.com.	
46:				TmBP/=s4hRvEvbLa...	
47:	servmail	IN	A	131.87.24.4	
48:		IN	SIG	A 1 86400	(
49:				19991023133034 19990923133044 59104 mydomain.com.	
50:		IN	NXT	mydomain.com.	A SIG NXT
51:		IN	SIG	NXT 1 86400	(
52:				19991023133034 19990923133044 59104 mydomain.com.	
53:				Al0Va/fuT23mRs...	)

Figure 3. The same zone file in DNSSEC

main, that is with the private key of *com*. To verify it, you must retrieve also the public key of *com*. In the same manner, the public key of *com* is signed by the root private key. After verification of the public key of *com* (with the public key of the root that you have), you reached a point in the tree that you can trust. Therefore, you can conclude that the public KEY of *mydomain.com* can be trusted. In this way, a resolver would learn trusted keys upon verifying their signatures passing through this chain of KEY and SIG RRs. The second time it needs to verify whether or not to trust a key, it can use the data cached during previous validations. In the worst case, a resolver would have to confirm the signatures of keys up to the root level of the DNS tree.

C. DNS Transaction Security

This new notion was introduced in the DNSSEC terminology due to several facts that concern the resolvers. A resolver is usually a simple application, that is not capable of caching and does little processing. In the general

DNSSEC scheme, if a resolver has to verify all the signatures of the RRs received passing through the chain of KEYs and SIGs described first it would have to be adjusted with cache capability and with the ability of verifying signatures. By far this solution is impractical since a lot of resolvers work on slow machines and also they interact with only a small number of name servers. A better solution is to let the task of verifying the signatures to the name servers and to introduce communication security between the name servers and resolvers.

For only two participants we can use secret key cryptography with the mentions that secret key cryptography is faster than public key cryptography and requires less CPU usage, so the load on resolvers would not be high. In this scenario, the resolver and the name server share a secret key. The key has to be generated out of band. This key - the TKEY meta RR - is not stored or cached in the DNS and should not appear in the zone files. The key can be generated both by server or resolver, hence we can have server assigned or resolver assigned keying. For example, in the case of a server assigned keying, the DNS server produces the keying material. The resolver sends a query in which it asks for a TKEY RR. In the additional section of this query the resolver includes its public key, that will be used by the name server to encrypt the keying material and compile the response for the resolver. Only the resolver can decrypt the symmetric key since it has the private key. The keying material will usually be less than 256 bits because that is enough at the present time for strong protection with modern keyed hash or symmetric algorithms. Another distribution method such as manual key exchange is possible.

At this point, the resolver and name server have a shared secret key. From now on, every message from the resolver can include a request signature and every message from the name server to the resolver can incorporate in the additional section a transaction signature. These signatures are created with the shared symmetric key and are authenticated by the receiver. For these special signatures, a new resource record is introduced - TSIG (Transaction Signature). The difference between SIG and TSIG RR consists in the fact that the first are signatures of sets of resource records while the second are signatures of DNS messages (for the resource records in the message and the header section).

The only message digest algorithm to be used for transaction signatures specified in the Internet draft [10] is HMAC-MD5 (as defined in RFC 1321, RFC 2104).

D. DNS as a PKI

We discussed the capability of DNSSEC to store public keys in KEY RR. But these RRs can store more than just DNS public keys. KEY RRs are associated to zones (used to sign DNSSEC zones), to hosts or end entities or to users (DNS can store user names). Additionally, each KEY RR is associated to a protocol, e.g. DNSSEC, IPSEC, a.s.o.

With the omnipresence of DNS in the Internet, this feature of storing keys can be used by other applications and protocols as a Public Key Infrastructure (PKI).

In the existing public key infrastructures, the public keys are published and authenticated by means of certificates. A certificate is a set containing a cryptographic public key, a validity interval, identity, and other related information all tied together by a digital signature. Additionally, a certificate revocation list represents the list of certificates that are revoked, all signed by the issuer of the revoked certificates. E.g., the X.509 certificates and their related CRLs in the X.500 directory or the PGP certificates/revocations.

The DNSSEC “chain of trust” provides some sort of certification since the verification of KEY and SIG RRs is similar to the verification process of a certificate in a PKI (Public Key Infrastructure). Moreover, in the RFC 2538 [7], a new resource record is defined for DNS to provide storage for certificates and their related certificate revocation lists - the CERT RR. According to the RFC [7], the certificates are recommended to be stored under a domain name related to the entity that controls the private key corresponding to the certified public key. Also, the CRL CERT RRs are recommended to be stored under a domain name related to the issuer of the revoked certificates.

Current DNS implementations are optimized for small transfers, typically no more than 512 bytes. Therefore, at the present time the RFC 2538 [7] recommends that is advisable to make efforts to minimize the size of certificates stored within the DNS. Efforts are also made to achieve efficiency for large transfers in the next generation of DNS implementations. For this, solutions may include the use of fewest possible optional or extensions fields and also the use of short field values for variable length fields that must be included.

E. DNSSEC - State of the Art

By the time of writing this article, the only implementation of a DNS secure is offered by the Trusted Information Systems (TIS) and has as foundation the BIND ver. 4.9.4. The secure DNS prototype from TIS supports only the KEY, SIG and NXT RRs. It also provides a signing tool based on the RSAREF cryptographic library. The TIS labs are also involved in developing the implementation of

DNS security to be used in the future versions of BIND. The work is funded by the U.S. Department of Defense's Information Systems Agency (DIAS).

On the other hand, the new version of BIND - BIND ver. 9 - is meant to provide full support for DNS security. The release of BINDv9 is due for April 2000. As their implementers say, BINDv9 is a major rewrite of the underlying BIND architecture in order to provide scalability, security (support for DNSSEC, support for TSIG), maintainability, portability and IPv6 support.

V. Observations

In the perspective of Domain Name System, the cryptography is used only for authentication and not for confidentiality. This is necessary for the resolvers to retrieve verifiable correct information from DNS servers out there. It also respects the primary goal of DNS to provide the same, public available, answers for all queries without discrimination.

The DNSSEC involves cryptographic keys and therefore some attention should be paid regarding key generation, key size, key storage and key lifetime problems. On the other hand, it is obvious that the KEY and SIG RRs are larger than any other resource records in DNS. If we compare a 4 byte address resource record with the at least 128 bytes of a SIG RR generated by a 1024-bit RSA private key, it is clear that the first remarkable difference between DNS and DNSSEC is represented by a substantial increase in size of the zone files and consequently of the DNS messages exchanged. As seen in our example, each name with a set of small-sized resource records from an unsecure zone file will be attached additionally, in a secured zone file, with a SIG and a NXT RRs.

Returning to key handling in DNSSEC, it is useful to mention a few things. Careful generation of key is of most interest and should not be ignored. The strongest algorithm used with the longest keys are of no use if a potential adversary can guess enough to reduce the size of the key space so that a powerful machine can exhaustively search it.

Another aspect considered is the lifetime of a pair of public keys. The longer a key is in use the greater the possibility of being compromised through carelessness, accident, espionage, or cryptanalysis. The RFC 2541 [9] recommends that no long term key should have a lifetime bigger than 4 years and a reasonable value for such keys would be represented by 13 months, with the idea that the keys be replaced each year. On the other hand, public keys with a too short lifetime can lead to excessive resource consumption in re-signing the zone data and retrieving fresh information because the cached information has expired. The minimum value proposed is 3 minutes - the reasonable estimate of the packet delay. In what concerns the lifetime of the keys used for transaction security the advisable time interval is 36 days with the intent that the keys should be changed monthly.

As computers get faster each year, cryptanalysis becomes more and more efficient in breaking keys. To combat this, it would be good to increase the size of the keys to a point where for the present computer's computation power it would necessitate a large period of time for breaking it. The problem is larger keys are indeed more secure but also slower. Moreover, larger keys imply growth of the KEY and SIG RRs, and also of the DNS message size. Therefore, it gives the possibility of DNS UDP packet overflow and hence TCP protocol would have to be used with its higher overhead. All this can only drive to slower name resolution with all its consequences, see RFC 2539 [8].

Equally important is the storage of private keys. The recommendations for this matter are that the zone private keys and the zone file master copy be kept and used for signing off-line, on non-networked and physically secure computers. Also, the secure resolvers must be configured with some trusted on-line public key information or they will be unable to verify the signatures of the resource records retrieved. Moreover, this public key must be protected too, otherwise it is possible that spoofed DNS information may appear authentic. The other type of private keys such as host or user keys, generally have to be kept on-line, since they might be used for transaction security key establishing.

For the top-level domain zones and for the root zone the problem of securing the private keys must be discussed in a different way. An attacker who could get the private key of a top-level zone would become authoritative for all the subdomains below. In the same way anyone who could obtain the root zone private key would be in control over the entire DNS space of all the resolvers configured to use the public key of the root zone, excepting those that are configured with the public key of a subdomain they belong to. Hence, the security of the root zone and top-level zones private keys is of major importance. The strongest, largest size, and most carefully handled keys should be used for these zones and the root zone private key should always be kept off-line. The lifetime of such a key should be of ten to fifteen years since the update of a huge mass of resolvers around the Internet would be difficult to achieve oftenly (also all the top-level public keys would have to be signed again by the new private key of the root zone).

VI. Conclusions

With this article we wanted to provide an introduction to the DNS security extensions pointing out the weaknesses of the present DNS implementations and also a guide to its secure extension DNSSEC. We found difficult starting work on this topics due to the lack of organized documentation on existing implementation of DNSSEC and to the fact that not all the security extensions mentioned earlier are integrated in the existing prototypes. The future work will be directed toward installation and testing of the new DNS implementations (that include the DNSSEC subset). Another aspect that will be analyzed will be the possibility of integrating DNSSEC in existing applications particularly focussing on PKI-based applications.

References

[1] RFC 1034 *"Domain Name System - Concepts and Facilities"*, Paul Mockapetris, ISI, November 1987.

[2] RFC 1035 *"Domain Name System - Implementation and Specification"*, Paul Mockapetris, ISI, November 1987.

[3] RFC 2065 *"Domain Name System Security Extensions"*, Donald Eastlake, IBM, C. Kaufman, January 1997.

[4] RFC 2535 *"Domain Name System Security Extensions"*, Donald Eastlake, IBM, March 1999.

[5] RFC 2536 *"DSA KEYs and SIGs in the Domain Name System (DNS)"*, Donald Eastlake, IBM, March 1999.

[6] RFC 2537 *"RSA/MD5 KEYs and SIGs in the Domain Name System (DNS)"*, Donald Eastlake, IBM, March 1999.

[7] RFC 2538 *"Storing Certificates in the Domain Name System"*, Donald Eastlake, IBM, Olafur Gudmundsson, TIS Labs, March 1999.

[8] RFC 2539 *"Storage of Diffie-Hellman Keys in the Domain Name System (DNS)"*, Donald Eastlake, IBM, March 1999.

[9] RFC 2541 *"DNS Security Operational Considerations"*, Donald Eastlake, IBM, March 1999.

[10] Internet Draft *"Secret Key Transaction Signatures for DNS (TSIG)"*, Paul Vixie (Ed.) (ISC), Olafur Gudmundsson (NAILabs), Donald Eastlake (IBM), Brian Wellington (NAILabs), July 1999.

[11] Brian Wellington, *"An Introduction to Domain Name System Security"*, TIS Labs, January 1999.

[12] Paul Albitz, Cricket Liu, *"DNS and BIND"*, Third Edition, O'Reilly, Sebastopol, CA, 1998, ISBN 1-56592-512-2

Appendix

I. Introduction to DNS

A. A Bit of History

DNS is the shorthand for the Domain Name System. It represents the set of protocols and services on a TCP/IP network which allow users of the network to use hierarchical user-friendly names when looking for other hosts instead of having to remember and use their IP addresses. This system is used almost by any other application and protocol that is involved in network communication (e.g., web browsing, ftp, telnet or other TCP/IP utilities on Internet).

At the beginning of Internet, the name resolution was performed by means of *"hosts"* files (e.g., */etc/hosts* in UNIX) which contained the complete list of names and their associated IP addresses. These files were administered centrally, by the Network Information Center (NIC), and each computer connected to the Internet had to update its file periodically. With the exponential growth of the Internet, this became a burden for system administrators, so a better solution was needed. And it was given by prof. Paul Mockapetris the main designer of the Domain Name System.

So, the best known function of DNS consists in mapping symbolic names to IP addresses and viceversa. One example, if we need to connect to a certain web site, we need to know the IP address of the machine that supports this service, (for example, something like this 131.87.24.29), instead of this sequence of ciphers, not so easy to remember and use, we could use the more suggestive name *www.mydomain.com*. This is where DNS gets involved.

In the ISO/OSI hierarchy, DNS finds itself on the application level, even though its usage is transparent to the users that simply refers to names instead of IP addresses, and it can use either TCP or UDP as transport protocols. Usually, the resolvers are mainly relying on UDP (since the DNS queries and responses are well-suited for this protocol), but TCP might be used whenever truncation of the returned data occurs.

B. DNS Overview

In practice, the Domain Name System can be seen as a distributed database of names. These names establish a logical tree structure called *domain name space*. The root of the tree is the root domain followed by its children, the Top Level Domains (TLDs), which in turn can contain several levels of subdomains. Figure 4 shows the structure of such a tree.

Host names consist of concatenation of the labels of each node on the path from the leaf that represents the ac-

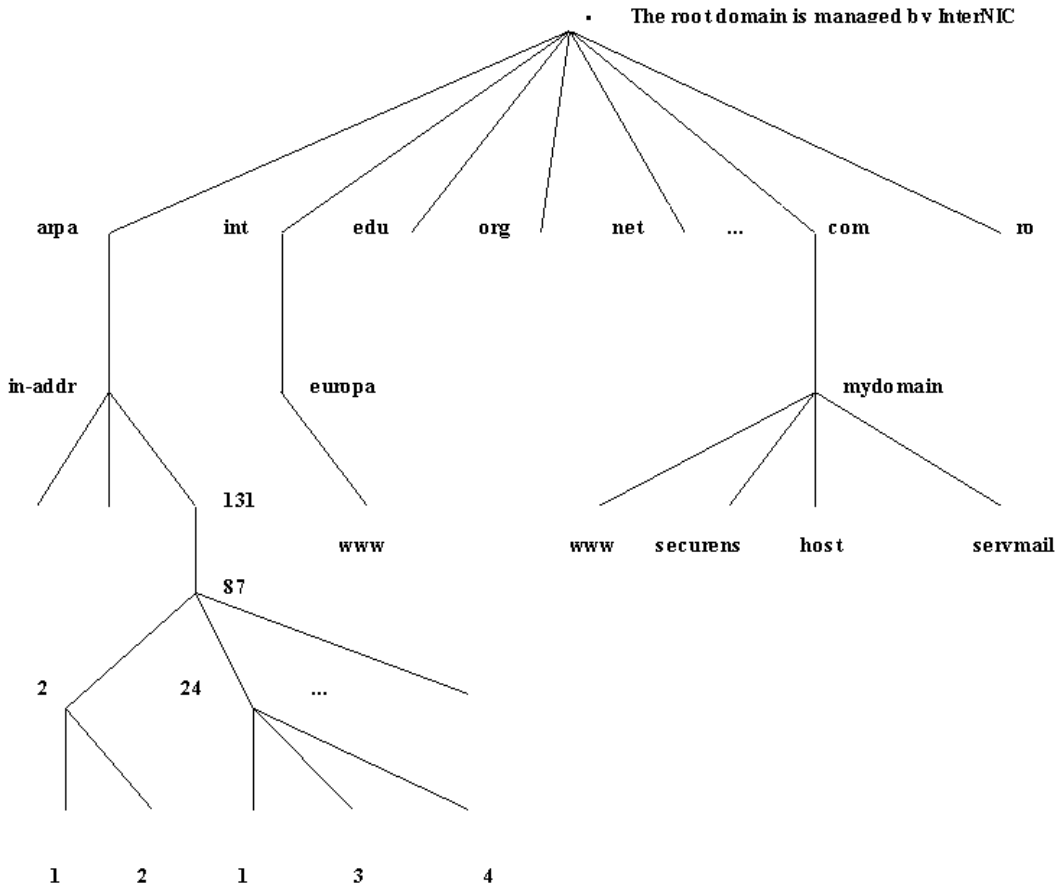

Figure 4. The Domain Name Space

tual host up to the root. Adjacent labels are separated by a dot. A general rule for host names would look like this *host.subdomain.top-level-domain.*

The DNS has three major components:

- The first category contains:
  - the **Domain Name Space** and
  - the **Resource Records**, that are specifications for a tree structured name space and the data associated with these names.
- **Name Servers** are server programs which maintain the information about the DNS tree structure and can set information. A name server may cache information about any part of the domain tree, but in general it has complete information about a specific part of the DNS. This means the name server has authority for that subdomain of the name space - therefore it will be called *authoritative*.
- **Resolvers** are programs that extract the information from name servers in response to client requests.

Each node in the tree of DNS database, along with all the nodes below it, is called a *domain*, that means a logical subtree of the domain name space. Each node or domain can contain subdomains. Domains and subdomains are grouped into zones to allow for distributed administration of the domain name space. Therefore, the portion of the name space whose database records exist and are managed in a particular zone file is called a *zone*. Name servers generally have complete information about some part of the domain name space (that is, a *zone*), which they load from a file or from another name server [12]. It is important to understand the difference between a zone and a domain. One of the main goals of DNS is to decentralize administration and this is realized by *delegation*. Therefore, an organization administering a domain can divide it into subdomains. Each of these subdomains can be delegated to other organizations. This means that an organization becomes responsible for maintaining all the data in that subdomain. It can freely change the data and even divide its subdomain up into more subdomains and delegate

	DNS QUERY	DNS RESPONSE
HEADER	qr, rd	qr, rd, ra
QUESTION SECTION	mydomain.com, type=NS, class=IN	mydomain.com, type=NS, class=IN
ANSWER SECTION	EMPTY	mydomain.com. 20h55m21s IN NS secuxens.mydomain.com. mydomain.com. 20h55m21s IN NS ns.otherdomain.com.
AUTHORITY SECTION	EMPTY	mydomain.com. 20h55m21s IN NS secuxens.mydomain.com. mydomain.com. 20h55m21s IN NS ns.otherdomain.com.
ADDITIONAL SECTION	EMPTY	secuxens.mydomain.com. 23h44m4s IN A 131.87.24.1 ns.otherdomain.com. 23h33m28s IN A 131.87.2.2

Figure 5. Example of DNS message exchange

those, too. All top-level domains, and many domains at the second level and lower are broken into smaller, more manageable units by delegation. These smaller units are the zones. A zone contains the domain names that the domain with the same domain name contains, except for domain names in delegated subdomains [12]. The zone data is stored in the zone data files which are loaded by the primary name servers authoritative for that zone.

The DNS messages are the data units that are transmitted between name servers and resolvers. The message format (see figure 5) consists of a header, containing a number of fixed fields that are always present, and up to four sections that carry query parameters and resource records. Data that is associated with the nodes and leaves of the DNS tree is exchanged in the last three sections of the DNS message. These resource records (RR) are labeled according to the type of data they contain. They may represent a host address (A), a name server (NS), a start of authority (SOA), a pointer to another location (PTR), an alias (CNAME), a mail exchanger (MX) or other types as specified in RFC 1035 [2]. The content of these four sections serves different purposes. Their order is always the same and some of them can be empty. The answer section, the authority section and the additional section have the same format.

- The *header* describes whether the message is a query or a response (with the *qr* bit), the type of the query (described by the 4-bit OPCODE field), an authoritative answer (AA), a truncated message (TC), recursion desired (RD), recursion available (RA), the response code (no error, format error, server failure, name error), etc.
- The *question* section carries the query name, the query type and the query class (e.g., IN, CH and HS standing for INTERNET, CHAOS and HESIOD classes). The most frequently used class is the INTERNET class. Valid query types are all the codes for resource record types (e.g., NS, A, MX, PTR, etc.)
- The *answer* section carries the resource records that directly respond to the query.
- The *authority* section contains resource records that describe other authoritative servers.
- The *additional* section carries the resource records that are not explicitely requested, but might be helpful in using the resource records in the other sections.

The whole database is divided into zones that are distributed among the name servers. The essential task of a name server is to answer queries using data in its zone. To ensure a higher degree of reliability of the system, the definition of DNS requires at least two name servers containing authoritative data for a given zone. The main name server is called the *primary* name server and the backup servers are called *secondary* name servers. Secondary authoritative name servers update their zone periodically with the data polled from their primary servers. Primary name servers load the data base files provided by the zone administrator and maintain a cache of data that was acquired through resource records. Since servers want to update dynamically the changes in the name space of their authorities, each resource record will contain a *Time To Live* (TTL) field to ensure that servers will not cache data beyond this time limit.

As shown in figure 6, the interface between the Domain Name System and the user programs is the *name resolver*. The resolver is on the same host as the user program and can contact one or more name servers. The resolver has a triple functionality: translating host names to IP ad-

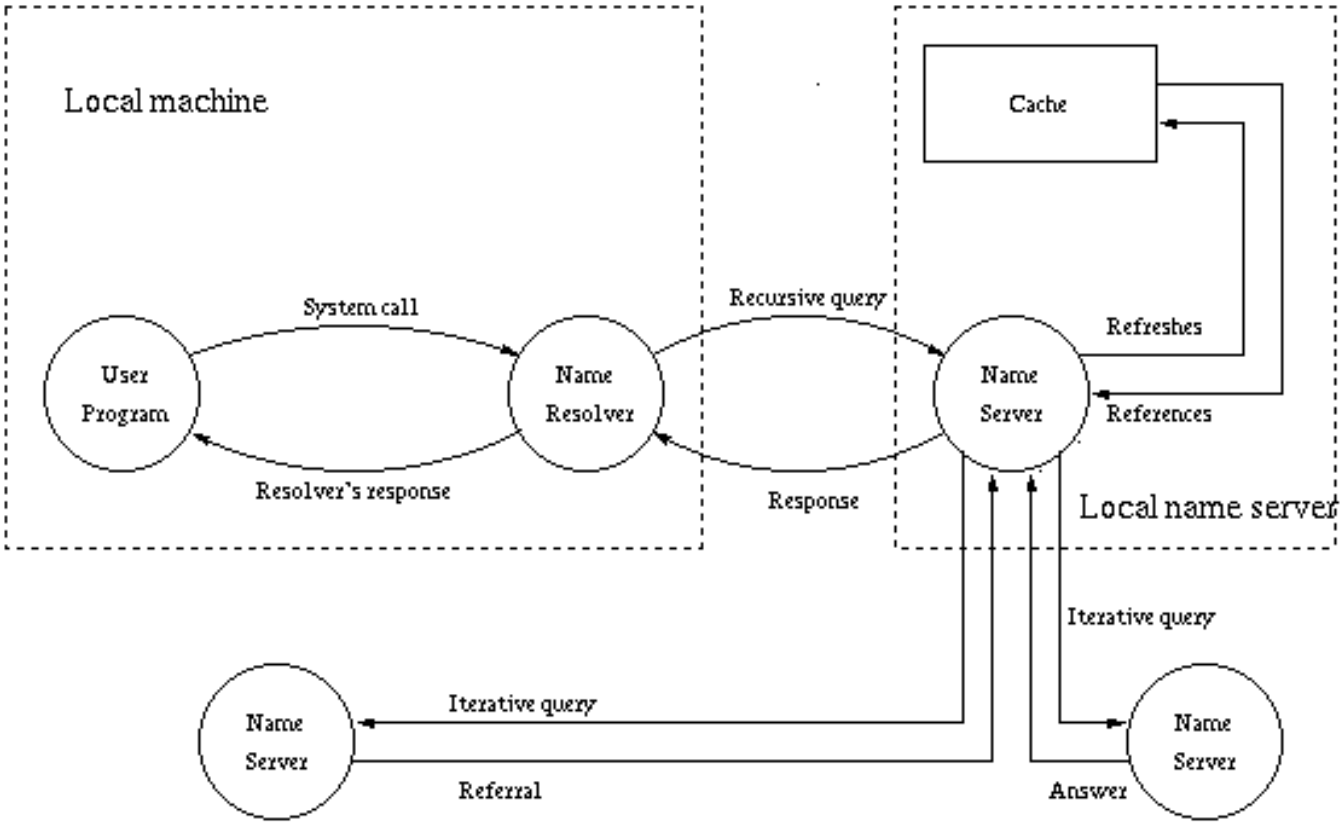

Figure 6. Interaction between resolver and name servers

dresses, translating IP addresses to host names and lookups for other information about domain names (i.e., resource records specified by type, class and name). When a user program needs to resolve a name, it addresses the resolver. The resolver in turn carries out the following procedure:

- It formulates a query and contacts its local name server which is pre-configured (e.g., in the *resolv.conf* file). This query can be either iterative or recursive.

– In *recursive resolution*, the queried local name server has to answer either with response to that query or with an error code. The queried local name server cannot refer to another name server. In case the local name server is not authoritative for the requested data, it has to resolve the query again for the resolver: iterative or recursive. After possible many referrals, the local name server finally finds the authoritative name server which will return either an answer or an error code. This information is then passed to the resolver.

– In *iterative resolution*, the local name server simply returns the best answer it has. This means either a response for the query or a referral to another name server that would help the resolver continue the resolution process.

The resolvers may vary from simple system calls (such as `gethostbyname()` or `gethostbyaddress()`) to more complex applications capable of caching information (e.g., *nslookup*, *dig*, *host*, a.s.o.). For efficiency reasons, the recursive resolution is the most usual type of DNS query between resolvers and the local name server so that all burden of the resolution process is carried out by the local name server. The resolution process seems complex and twisted compared to a simple lookup in a host database, but the speed-up is offered by caching. To answer a query a name server might need to send several DNS messages. During these resolution attempts the name server discovers new information about the name space that are stored in a local cache that will help speed-up the future queries. The next time the resolver queries the name server for the data about a certain domain name the name server knows about, the process is shortened significantly.

C. DNS - State of the Art

The most extensively used implementation of the Domain Name System is BIND. BIND stands for Berkeley Internet Domain Name; initially implemented as a graduate student project at the University of Berkeley, then for a few years it was developed by DEC and afterwards the task of developing BIND was taken over by the Internet

Software Consortium (ISC). The current available version is BIND 8.2.1. The future version BIND 9 is due to be ready in April 2000, but a beta version for testing will be made available in the first quarter of year 2000.

Another implementation to be mentioned is the Microsoft DNS. As the company says it is not a port of BIND code, but rather a rewrite - RFC compliant - of it. It is included in the Windows NT Resource Kit and is available for platforms running Windows NT 4.0 and upper versions (also available for Windows 2000).