Enhancing DNS Security using the SSL Trust Infrastructure

Christof Fetzer and Gert Pfeifer
Dresden University of Technology,
01069 Dresden,
Germany
{christof.fetzer, gert.pfeifer}@inf.tu-dresden.de

Trevor Jim
AT&T Labs-Research,
180 Park Ave.,
Florham Park,
NJ, 07932, USA
trevor@research.att.com

Abstract

The main functionality of the Domain Name System (DNS) is to translate symbolic names into IP addresses. Due to the criticality of DNS for the proper functioning of the Internet, many improvements have been proposed for DNS in terms of security and dependability. However, the current secure DNS (DNSSEC) standard has still several problems that need further consideration. For example, online updates and denial of service attacks are not sufficiently addressed. These problems are serious obstacles that might prevent DNSSEC from replacing the traditional DNS. In this paper we discuss several of these technical and economic problems. To address these issues, we propose a simple extension to the existing DNS. It is SSL based and individual domains can decide independently of each other if and when to adopt the extensions. We show how to implement these extensions with the help of a simple proxy DNS server. **Keywords:** network services, security, WWW, DNS, dependable systems, distributed systems

1 Introduction

The Domain Name System (DNS) is used to map symbolic names onto IP addresses. Many Internet applications rely on this service, which has been available for over 20 years. During this time many vulnerabilities have been found in DNS, e.g., denial of service [4] and cache poisoning [7]. Cache poisoning happens when an adversary inserts bogus information into a DNS cache. When a DNS server A queries the DNS server of an adversary B, B can add bogus information to its reply message. A uses the content of the message to update its cache and in this case, is poisoning its cache. The fundamental problem is that the correctness of the received information cannot be verified because the DNS entries are not authenticated. To address this problem, a new standard for a secure name service (DNSSEC) has been developed. Unlike DNS, DNSSEC authenticates DNS entries. However, after more than ten years of development, DNSSEC has still not been deployed on a large scale.

We believe that there exist three major issues that have slowed the deployment of DNSSEC. First, the migration from DNS to DNSSEC is difficult and costly. Second, some companies might have vested interests in delaying the roll out of DNSSEC (see Section 3.3). Third, the trust model of DNSSEC does not seem to be a good match with the end-to-end SSL trust model which is widely used by Internet applications.We propose an approach to address these problems.

The main advantages of our proposed approach are: (1) ease of implementation, (2) ease of use, and (3) an inexpensive roll out. Our ap-

proach is based on the the existing DNS infrastructure extended with SSL technology. Our approach is evolutionary and provides a smooth migration and roll-out. An entity that needs the additional security can decide to adopt our extensions independently of other entities. While some stakeholders might also want to slow down the adoption of our extensions, the autonomy of adoption can restrict the influence of such stakeholders.

The remainder of this paper is organized as follows: Section 2 outlines the major security problems of DNS and Section 3 discusses how DNSSEC solves them, or not. Furthermore, we describe how DNSSEC competes with other technologies and how different organizations and their interests are involved in this competition. Section 4 explains how an alternative security concept could be realized with the help of DNS. Section 5 concludes the paper. A glossary can be found in the Appendix.

2 DNS Security Issues

DNS was originally specified in 1984 by Paul Mockapetris in RFCs 882 and 883. It was described as a distributed database with the purpose to replace the HOSTS.TXT file that was used to map hostnames to IP addresses and vice versa.

The world wide web was growing very quickly particularly in the late 90's during the e-commerce and telecommunication boom. But already in the mid 80's HOSTS.TXT was no longer a suitable solution for naming in the Internet.

DNS, the successor of HOSTS.TXT, is much more scalable. The hierarchical system of DNS names allows local control of segments of the overall database, called zones. Each DNS zone takes responsibility for entries in its local zone.

2.1 Trust Model

Resolving symbolic names typically requires DNS requests to be sent to a sequence of name servers. Most operating systems use however very simple stub resolvers which are not able to perform such recursive resolutions of names (nor a caching of results). A local DNS server is typically responsible for performing the recursive resolution of symbolic names on behalf of the stub resolvers.

The correct resolution of a symbolic name requires that correct information be provided by all DNS servers involved in a recursive query. Because the communication between DNS servers is neither encrypted nor authenticated, the correctness also depends upon the assumption that information is not modified by the network. While the latter could be enforced be using VPN technology[1] for the communication channels between DNS servers, using a VPN would not prevent DNS servers from sending wrong information to other DNS servers. The basic reason for this is that DNS entries are not authenticated and hence, detecting whether entries have been modified is difficult or impossible.

DNS supports online update because many DNS entries depend on the assignments of DHCP servers, but it does not check the integrity of these updates.

These scenarios demonstrate that DNS depends on the assumption that all DNS servers and the communication channels between the DNS servers are well behaved. Therefore, a critical application should not rely on the assumption that a host name is properly resolved by DNS to determine the authenticity of a host. Instead, critical applications need to perform an application level end-to-end authentication.

DNS has the following major security prob-

[1]The TSIG extension, introduced in RFC 2845, is not suitable in this case. TSIG can be used if there are only few authorized clients, since it is based on shared secrets. In a larger scenario there is a need for public key algorithms and trusted third parties.

lems:

- Denial of service attacks on DNS [4] are helped by the following two properties. First, there are only a few root DNS servers. Second, DNS queries are much smaller than the corresponding server responses.
- DNS entries are not authenticated. That means that a user is not able to verify whether a particular entry has been forged or not.
- DNS client/server communication is not protected. Some adversary could forge requests or responses on the way.
- DNS servers do not authenticate themselves. Some adversary could pretend to be some valid DNS server.

2.2 Trusted Hosts Mechanism

There are quite a few articles focusing on security issues of DNS. An early article by S. Bellovin [3] points out the problems of the concept of *trusted hosts* which is used in connections with DNS. *Trusted hosts* is a very simple mechanism that is used by a couple of remote tools like *rlogin*, *rcp* or *rsh*. When one of these *r*-commands is used between two hosts, they must trust each other via /etc/hosts.equiv[2] files. This raises four major problems:

1. Some adversary might forge his IP address to get the permission to use a remote command or tool on the victim's host. This class of attacks is called spoofing and is not within the scope of this or Bellovin's paper since it does not concern DNS directly.
2. Some adversary could prevent users to use *r*-commands using DNS vulnerabilities. Enabled by the circumstance that DNS queries and responses are not authenticated and not encrypted they can easily be manipulated.
3. Some adversary could forge DNS entries on the DNS server itself. These entries are not protected by any authentication method.
4. To accept an incoming connection the server must check whether the client is a trusted host. To find this information in the /etc/hosts.equiv it first has to look up the hostname because it only can see the IP address within the source field of the incoming IP packets resp. their headers. This can be done with a reverse DNS lookup. Inverse mappings are implemented by a separate DNS tree, keyed by IP addresses. The fundamental flaw that Bellovin exploited for his attack was that there is no forced linkage between the two DNS trees. The forward mapping and the inverse mapping can be managed by different DNS servers with different locations and different managers.

Bellovin's paper was withheld from publication for 5 years. It took some time to find feasible fixes for the vulnerabilities exposed in the paper. In the end Bellovin agrees with many other authors that not the lack of authentication in DNS is the reason for the vulnerabilities but the authentication method used by those applications.

This problem is still present. Of course we are now using ssh instead of rlogin, but we still trust URLs in unauthenticated connections like in our web browsers. Also, some services (e.g., online libraries) use now IP addresses instead of host names for authentication. However, these address based authentication mechanisms are often circumvented using a public proxy server in the proper IP address range.

[2] This file contains a list of trusted users and hosts. It allows these users to use tools like rexec.

3 DNSSEC

On the 28th IETF meeting in Houston in 1993 the starting shot for the organized work on DNSSEC was given. The design team meeting summary was later posted to the dns-security@tis.com mailing list. Relating security issues there were just two important requirements:

1. data integrity
2. data origin authentication

But there were some further important decisions:

- no encryption: DNS data is public data
- no authentication of clients and servers
- backwards compatibility and co-existence with DNS

These requirements were also the starting point for our approach and, as we will explain in Section 4.2, our approach complies with these conditions better than DNSSEC. The DNSSEC specification seems to approach a final state now, more than ten years after the initial meeting for DNSSEC.

3.1 Security Issues

There are a few papers that discuss the weaknesses of DNSSEC. For example, in April 2004 Derek Atkins and Rob Austein wrote an Internet-Draft [2] that describes security problems of the current DNSSEC specification. The main points of their critique are:

- DNSSEC servers are more effective as denial of service amplifiers
- answer validation increases the resolvers work load
- the trust model is almost totally hierarchical
- the implementation is very complex
- key roll-over at the root zone is really hard
- requirement of loose time synchronization between server and resolver
- inherent complexity of the wildcard mechanism complicates the authenticated denial mechanism (i.e., the proof that a node with a given name does not exist).

Besides those technical issues there are further concerns that bar DNSSEC from being deployed quickly as pointed out in the following subsections.

3.2 Economic Aspects

A problem that was not mentioned yet is the economic aspect of DNSSEC As a matter of fact there are many more hostnames and hosts with low security requirements than hosts with high security requirements. If DNSSEC is used to protect them, the cost to benefit ratio is not satisfactory. DNSSEC can only be used to protect the whole zone and there is hierarchical trust. That means that the higher zones up to the root zone must be protected too. So this is an approach that needs a consensus. We can assume that users would not pay an additional fee for security. The reason is that the majority of them do not need additional security.

The DNSSEC benefits must justify large investments required for its roll out. This could be reached by high consumer or industry demand, but it seems that those players who need the security did not wait ten years for DNSSEC. They found other suitable solutions. In Section 3.3 and 4 we show how a higher level of security could be reached without DNSSEC.

3.3 The competition

SSL is a tunneling protocol that provides authenticated and encrypted sessions between

servers and clients. SSL starts with a handshake that first establishes a TCP/IP connection. This means that SSL is located above TCP in the protocol stack. It is not suitable for protecting UDP communication. The participants are authenticated using public key cryptography. Once authenticated, they select the strongest cryptographic algorithm supported by both. SSL can use a couple of so called cipher suites, that include several checksum and encryption algorithms. SSL was developed by Netscape in the early 1990s. Version 1 was never released, a Version 2 client was included in the Netscape Navigator 1.1 in 1994. Version 3 is available since 1995.

At first glance, it is not evident that SSL competes with DNSSEC, since SSL does not check the integrity of DNS mappings. However, SSL provides a method to verify the address obtained from DNS for a given host. SSL can tell the client that at least one trusted third party thinks that the host reached using this address is in fact authentic. This does not exactly match the semantics of DNSSEC, but the majority of the users or customers would not worry about that or would even prefer the end-to-end semantics of SSL.

SSL is used in exactly those applications where DNSSEC would be most valuable. In addition, SSL has several advantages over DNSSEC:

- SSL is mature and is well investigated.
- There is a kind of brand awareness. Browsers display a lock and the URL prefix "https" for SSL connections, and users have the idea that this indicates a secure connection.
- SSL is supported by many software products like operating systems, web servers, and client programs for different purposes.
- SSL requires little or no further development.
- SSL provides additional services: application-level authentication and encryption.

In short, SSL is a serious competitor to DNSSEC. It has taken the most valuable part of the market. Thus the need and the demand for DNSSEC has been reduced.

There are more DNSSEC competitors than SSL, e.g., many public key infrastructures could also be considered as competitors. All of them rely on the existing DNS and, like SSL, perform additional checks to authenticate their counterpart.

4 The Proxy Approach

To address the weaknesses of DNSSEC, in particular, the roll-out issues of DNSSEC, we propose a new approach, that

- needs a small initial investment.
- is evolutional, i.e., is based on existing technologies (BIND, SSL).
- does not touch todays stub resolvers, i.e. does not demand client updates.
- permits a simple implementation.
- provides a simple key roll-over.
- does not increase the resolver's workload.
- uses the existing SSL CA infrastructure.
- provides a simple online update mechanism for DHCP users.
- provides a simple heuristical denial mechanism (to witness that there is no entry matching on the given query)

The basic ideas of our proposal are presented in the following subsections.

4.1 Extending DNS

An easy way of enhancing the integrity of the DNS service is to add new resource record (RR) types and to use them for authenticated mappings. In the past, adding new RR types was difficult because it required changes to the server software[3]. Deployment of new server software is expensive and takes some time. It might be reasonable to change the software of an authorative name server in order to provide new security features for this zone. However, it does not seem reasonable to expect that the software of all DNS servers and clients will be changed if only a small fraction of users wants to add new security features.

If we want to include new RR types in order to enhance the integrity of DNS entries, we must make sure that all participants that are not aware of these records can deal with them transparently. BIND is the most frequently used DNS server implementation [6]. Since version 9.1, BIND includes experimental support for the transparent processing of unknown RR types without any additional recompilation. Our DNS security extension is based on this feature.

4.2 Owner-signed Resource Records

The most important problem of DNS RRs is, that they are not authenticated, i.e., do not have a signature. For example, if the CNAME RR would be replaced by an authenticated RR, let us call it SEC_CNAME, it would be extremely difficult for an adversary to forge such a RR—even if the communication between DNS servers is not protected.

A resolver with high security requirements could always ask for authenticated, i.e., signed, resource records. The signature would be created by the owner of the mapping. It is necessary to validate the certificate of the owner with help of a trusted third party, i.e., a certification authority. It is no longer necessary to trust all DNS servers because a client can verify an DNS entry owned by a zone A even if it was received from a DNS server of another zone B.

In our approach we plan to add signed entries for the following important DNS RR types: A, NS, PTR, CNAME, MX, AAAA and later also SOA.

Our approach fulfills the two major requirements mentioned in Section 3. The remaining three conditions are also fulfilled. Remarkable is, that our approach meets the 2^{nd} requirement in a better manner than DNSSEC does. The owner of a DNS entry is the entity that updates the record on the DNS server, not the server itself. When using DNSSEC, the zone administrator signs the zone data, although he did not necessarily create the entries, so this is not a real data origin authentication. In our approach we want the data to be signed be the entity that is responsible for it.

Determining the origin of a DNS entry is a new problem. To verify a signature one must find the certificate of the owner of the signed record. A simple solution could be that the owner is the entity, that owns a certificate containing a distinguished name that exactly matches the DNS name of the signed record. This approach is currently used by many web browsers to use https.

Example: The connection between a client and the host *www.hsbcprivatebank.com* is protected via *HTTPS*. The distinguished name of the SSL certificate is "www.hsbcprivatebank.com". The web browser accepts this certificate because the name of the host and the name in the certificate match. At the same time this certificate can be used to protect the integrity of the DNS mapping. This would not cause any additional cost because the certificate is needed anyway.

An important difference between our approach compared to DNSSEC is that the number of certificates and signatures in the system depends on the number of RR with high

[3] At least to relink or recompile.

security requirements. In DNSSEC there is one key pair per zone and each entry in the zone is signed. Our approach needs a certificate for each host with high security requirements. Only the entries of these hosts are signed. We expect that the number of signatures is much smaller in our approach and so there is a reduced demand for network bandwidth and memory.

To enable the resolver to verify a signed entry, the certificate of the signer must be sent as a kind of glue record. Today, the glue mechanism is used if a subzone is delegated to another name server and it is obvious that the resolver needs the IP of this server. This glue mechanism is also quite useful in our case, but there is a trade-off that needs further examination. A certificate can be quite large in comparison to a simple A RR. If the certificate is sent each time a resolver queries for a signed entry, this mechanism could be a denial of service amplifier. It might be better to let resolvers aggressively cache certificates instead.

4.3 Using SSL

Currently, we are using OpenSSL [1] to create and verify signatures. When a new secure RR is uploaded to the server the whole "insecure" entry is signed and inserted into the RDATA field of the new RR. The new record gets the original name, class and TTL. The type we are using in our prototype is from the private use realm as mentioned in [5]. The signature was created using the *rsautl* tool that comes with OpenSSL[4]. The certificate of the signer is not included in the signature. The original content is not encrypted.

If a secure RR has to be verified, OpenSSL needs the certificate of the signer, the signed RR and the certificate of the CA that signed the certificate. How the certificate can be acquired has been discussed in section 4.2. The CA certificate can be stored in a certain directory like https-aware Internet browsers usually do.

[4]S/MIME can also create signatures, but because of the encoding the signed message is much larger.

4.4 Online Update

Since the DNS server is not forced to sign entries, it does not need a private key for online operations. The owners of DNS entries have to supply signed entries. Sometimes these owners are forced to update DNS entries very often, e.g., if they are using DHCP. Such users need an mechanism to update signed DNS entries. There are two possibilities:

- The DHCP server can use or provide a service to sign entries on demand, or
- The DHCP server can select new signed entries from a pool of presigned ones.

The examination of these online update mechanisms is going to be documented in a future publication.

4.5 Denial of authenticated entries

If a resolver tries to get a signed DNS entry, the operation may fail if no such entry is available. The problem is to determine whether the authorative DNS server does not support secure entries or some adversary tries to prevent the resolver from accessing it.

In this case the resolver tries to find out whether the owner of the entry it looks for has published a certificate for use in DNS or not. These certificates may be published using services like *HTTP* or *NEWS* but also using a special DNS zone. If the resolver is able to find a certificate and the authorative name server negates its existence, the name server might not be authentic. If the resolver is not able to find an authorative name server, a denial of service attack is most likely. If no certificate is found, it must be assumed that the owner is unable to supply secure DNS entries.

4.6 Stub Resolvers

In many modern operating systems the DNS resolver is just a set of library routines like *gethostbyname(const char *name)* or *gethostbyaddr(const char *ip)* for reverse lookup. They are compiled into programs like mozilla or telnet. They are not even separate processes. Their functionality is restricted to sending requests, waiting for an answer and retry the query, if no answer has been received. Most of the work of finding an entry to the request is left to the server. Therefore the resolver performs a recursive query to the first name server (Figure 1). This server performs iterative non-recursive queries to other name servers.

This simple principle helps to keep the resolvers workload as low as possible. If the resolvers would have to evaluate certificates with cryptographic operations the whole stub resolver concept would be thrown overboard. But as the nearest DNS server can do the work of resolving DNS requests, it can also check their signatures. To permit this simplification, we assume that the client or stub resolver trusts the local DNS server and the connection between both is secure because it is, e.g., within an enterprise or virtual private network.

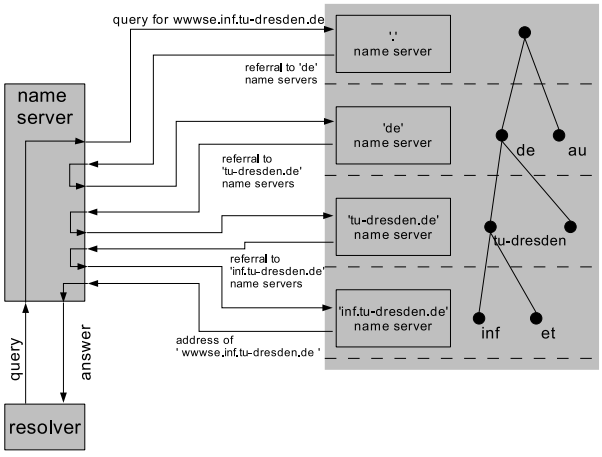

Figure 1: recursive resolution of local DNS server

4.7 Architecture and Algorithm

Our implementation idea makes the following assumptions:

1. All involved DNS servers can deal with unknown RR types transparently.
2. There is either a trusted connection to the local DNS server or certificates and signatures of secure DNS entries can be checked on the client's machine.
3. It is possible to find out whether a server supports our DNS extensions or not by looking for it's certificate(s) using external services like *NEWS* or *HTTP* is available. Another DNS zone would also be suitable.

The technical features of our approach are as follows:

- The stub resolver implementation is not touched.
- The DNS server implementation is not touched as long as requirement 1 holds (which is the case for the current BIND implementation).
- OpenSSL or another SSL implementation is used.

Further properties are:

- The DNS entry owner decides whether a secure entry is needed or not.
- The DNS client decides for which zones resp. hosts he only accepts secure, i.e., authenticated entries.
- The authentication is done by a proxy on a trusted server or on the user's machine. Of course, this will increase the workload on this machine.

The system works as follows (Figure 2): The stub resolver sends its requests to the DNS

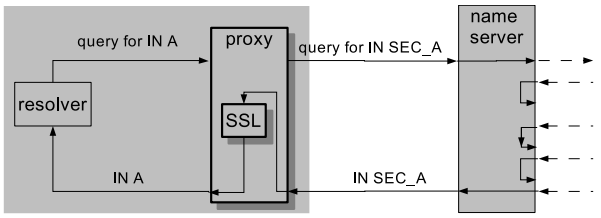

Figure 2: authentication via proxy

proxy. The proxy decides whether the requested entry needs authentication. For this purpose, it has a user supplied policy, e.g., in form of a list of zones or hosts with high security requirements. This list might for example contain the online banking server of the user. If the entry needs authentication, the proxy replaces the request type by the corresponding authenticated type, otherwise the original request is forwarded to the name server. After the name server answers to an "authenticated" DNS request, the proxy evaluates the signature and the certificate. It would be an advantage if the proxy maintained a certificate cache. If the entry is authentic, it sends a non-signed answer back to the stub resolver. Furthermore, the additional and authorative sections of the DNS message are forwarded to the stub resolver in the same manner. If the name server fails to find a signed entry, the proxy tries to find the certificate of the entry owner. This could be done with new special DNS zones. The authorative name server of such a zone provides certificates for all DNS owners that provide secure entries.

Requirement 2 of section 4.7 gives three possibilities for the placement of the proxy: 1. If there is a secure connection between the local DNS server, the proxy can be placed there. 2. If it is acceptable to have an increased workload on the resolver's machine, the proxy can be placed locally like depicted in Figure 2. 3. The proxy can be placed on any trusted host within a VPN.

5 Conclusion

We presented a new kind of DNS security extension that satisfy the original requirements for DNSSEC. The main competitor to our approach is DNSSEC. There are two kinds of advantages of our approach in comparison to DNSSEC.

Economical advantages: Our approach needs no further financial investment of DNS server providers. The only prerequisite is that the DNS server is able to deal with unknown RR types transparently. The roll-out of the security extension mechanism must be done by users that want to provide secure DNS mappings and those who want to use secure mappings. These are the people that really want the new features. Everybody else is not involved. In comparison with DNSSEC, this reduces and redistributes the overall costs.

Technical advantages: The size of the DNS database is inflated by DNSSEC. Our approach only increases the size of DNS entries of those hosts that really need the enhanced security. This is expected to be just a small fraction of the hosts listed in DNS. Hence, the impact on the zone file size is expected to be smaller. The workload of the resolver is not increased and the resolver implementation is not touched at all. Hence, do not need to be upgraded. There is no hierarchical trust model except possibly for the CA hierarchy. Our trust model provides end-to-end semantics with trusted third party involved. The implementation of our approach is very simple too. A resolver has to contact a proxy, that cares for the authentication or, if no authentication is needed, just forwards DNS requests. The key roll-over is very simple, since the owners signs their entries and they are able to update their entries in a simple manner. The denial mechanism (i.e., proof that a host name does not exist) is just heuristical and therefore very simple. The disadvantage is, that there is no proof for the existence or non-existence of a signed entry.

6 Future Work

We will investigate the following issues more closely: (1) heuristical denial mechanisms, (2) algorithms for determining the owner of a RR, (3) how delegations can be protected, (4) whether certificates should be sent as glue records or caching is a better strategy, and (5) the expected percentage of secure entries in DNS zone files.

References

[1] The OpenSSL Project. http://www.openssl.org/, 2004.

[2] D. Atkins and R. Austein. *Threat Analysis of the Domain Name System*. Network Working Group, 2004.

[3] S. M. Bellovin. Using the Domain Name System for system break-ins. In *Proceedings of the fifth Usenix UNIX Security Symposium*, pages 199–208, Salt Lake City, UT, June 1995.

[4] CAIDA. Nameserver DoS Attack October 2002. http://www.caida.org/projects/dns-analysis/oct02dos.xml, 2003.

[5] D. E. Eastlake, E. Brunner-Williams, and B. Manning. Domain Name System (DNS) IANA Considerations. RFC 2929, 2000.

[6] Internet Systems Consortium. ISC Internet Domain Survey. http://www.isc.org/index.pl?/ops/ds/, 2004.

[7] C. Irving. *The Achilles Heal of DNS*. SANS Institute, 2001.

A Glossary

A Record The resource record type A assigns an IP Version 4 address to a domain name.

BIND Berkeley Internet Name Daemon. The most common DNS software of the Internet.

CA Certification Authority

Certificate A trusted third party, i.e., a CA, confirms that a public key belongs to a person or organization by creating a certificate containing the public key, the name and some management information like expiration date and serial number. The trusted third party signs this structure.

CNAME Record Canonical Name Record. Represents an alias of a DNS name.

MX Record Mail Exchange Record. Represents a mail route for a domain name.

NS Record Name Server Record. An NS record declares that a given zone is served by a given name server.

PTR Record Pointer Record. Used to associate an IP address with a DNS name. This RR type is needed for reverse lookups.

RR The Resource Record is the basic unit of data in DNS. It contains five fields: Name (like www.cnn.com), TTL, CLASS (Internet, Hesiod, or Chaos), TYPE (A, MX, NS , ...) and the RData that contains the data of the record. The format of RDATA depends on the TYPE of the record.

SOA Record The Start Of Authority record is the first record in a zone file. It contains some information about the zone and tells the server that it is authorative for this zone, i.e., the zone contains it's own native data.

TTL Time To Live