12

SECURING DNS (PART I)

12.1 DNS VULNERABILITIES

As we've seen, DNS is fundamental to the usability of nearly every IP network application, from web browsing, email, to multimedia applications and more. An attack that renders the DNS service unavailable or which manipulates the integrity of the data contained within DNS can effectively render an application or network unreachable. Clearly protection of DNS data and DNS communications throughout the resolution process is critical. This chapter will focus on potential security vulnerabilities within DNS in general. Specific DNS server implementations may contain additional vulnerabilities, so as with any network service or application, monitoring of operating system, and associated software vulnerabilities is a fundamental operational process.

It's instructive in discussing DNS security vulnerabilities to consider the data sources and data flow within the DNS, depicted in Figure 12.1. Starting in the upper right-hand corner of the figure, DNS servers are initially configured with configuration and zone file information. This configuration step may be performed using a text editor or an IPAM system. For Microsoft implementations, the "IPAM system" may be the Microsoft Management Console (MMC). Configuration of server parameters and associated zones is required.

IP Address Management: Principles and Practice, by Timothy Rooney

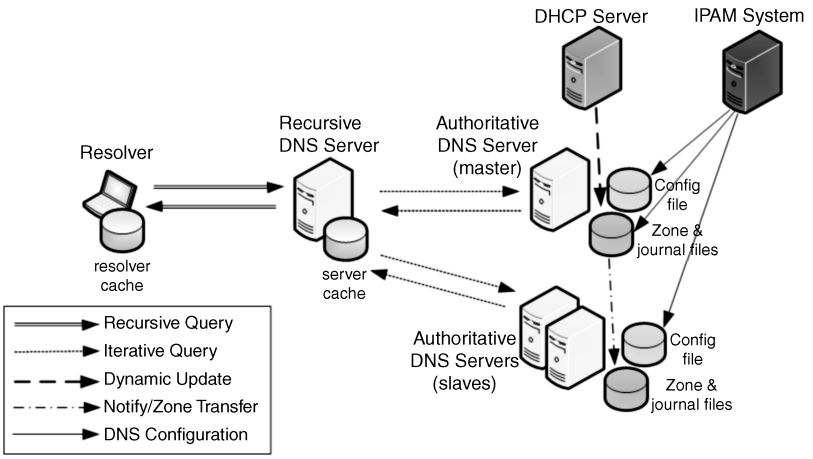

Figure 12.1. DNS data stores and update sources (11).

For BIND implementations, this configuration consists of a named.conf file and associated zone files on the master server. While a server may be master for some zones and slave for others, we'll use the master server terminology in assessing the vulnerability of a particular zone's information. Configuration of slave servers requires creation of the configuration file only, which defines the server's configuration parameters and its authority for particular zones. Slaves transfer zone information from corresponding masters.

Zone information may be updated by external sources as well, particularly DHCP servers. Dynamic updates can be accepted for clients obtaining dynamic IP addresses requiring DNS updating of address-to-name mappings. These updates will typically originate from the DHCP server assigning the address and will be directed to the server acting as master for the given zone. The master will add the update to its journal file and may then notify its slaves of the update, who may request an incremental zone transfer to capture the updated zone information.

Thus, authoritative zone information can be configured on a name server via zone file editing directly or by an IPAM system, and via zone transfers and dynamic updates, yielding several potential data sources and data update communications paths.

Beyond the configuration information and zone files, the third information respository within a DNS server is its cache. Cache information is accumulated through the query resolution process. As query answers are sought and received, corresponding answers are cached by the server. The cached information may be obtained not only from the Answer section of the DNS protocol message, but also from the Authority and Additional sections. These sections supply authoritative server information and information purportedly supplemental to the Answer. This information may include the authoritative servers for the relevant zone and other information related to the query (e.g., the A/AAAA "glue" record for an NS query).

The query resolution flow, beginning on the left of Figure 12.1, begins with the client resolver initiating a recursive query to its recursive server. Recall that the target server to

query is defined in the client's resolver configuration, managed manually or via DHCP. The recursive server will issue iterative queries as necessary through the domain tree to resolve the query, generally ending with a name server that is authoritative for the zone corresponding to the query. The master or any of the slaves are authoritative with the zone information. The authoritative server responds with the answer and potentially related information, for example, in the Additional section of the response. The recursive server will generally cache this information, as will the resolver. This cache will be relied upon for similar future queries to improve resolution performance. So it's important to assure data integrity of information returned to both the resolver and recursive server (i.e., both resolvers, the stub resolver in the client and the resolver within the recursive server).

Now let's look at the vulnerability of this information and communications model. RFC 3833 (139) thoroughly discusses various vulnerabilities to the DNS protocol and information integrity. We'll summarize those and some other vulnerabilities here, and then address mitigation strategies.

12.1.1 Resolution Attacks

- *Packet Interception or Spoofing*. Like other client/server applications, DNS is susceptible to "man-in-the-middle" attacks where an attacker responds to a DNS query with false or misleading additional information. The attacker spoofs the DNS server response, leading the client to resolve and cache this information. This can result in hijacking resolvers and hence applications to incorrect destinations, for example, web sites.
- *ID Guessing or Query Prediction*. Another form of malicious resolution is ID guessing. The ID field of the DNS packet header is 16 bits in length, as is the UDP packet header ID. If an attacker can provide a response with the correct ID field and UDP port number, the resolver will accept the response. This enables the attacker to provide falsified results, assuming the query type, class, and name are known or guessed by the attacker. This attack can potentially redirect the host to an illicit site. Guessing a 2^{32} number is relatively easy even with brute force methods.
- *Name Chaining or Cache Poisoning*. This packet interception style attack features an attacker providing supplemental resolution information usually within the Additional or even Authority section of the DNS response packet, thereby poisoning the cache with malicious query information. This may for example attempt to falsify information for a popular web site such as cnn.com, google.com or the like, so when such a query is requested, the resolver will rely on this falsified cached information. When the resolver is asked to resolve such a query, it will access its cache and utilize the malicious information to essentially redirect the client to the attacker's intended destination. An alternative approach to forcing the resolver to access the poisoned cache data is to provide an email link, which when followed, will resolve to the intended poisoned hostname. The so-called Kaminsky DNS vulnerability is a cache poisoning type of attack.

- *Resolver Configuration Attack*. The resolver on the client must be configured with at least one DNS server IP address to which DNS queries can be issued. This configuration may be performed manually by hard-coding the DNS server IP address in the TCP/IP stack, or automatically via DHCP or PPP. This type of attack may alternatively originate from an attacker launching it via a web plug-in, for example. This type of attack seeks to redirect the resolver to an attacker's DNS server to resolve to malicious data.

12.1.2 Configuration and Server Attacks

- *Dynamic Updates*. An attacker may attempt to inject or modify data in a DNS zone by attempting a dynamic update to the server. This type of attack attempts to redirect resolutions from clients for the intended destination to an attacker-specified destination.
- *Zone Transfers*. Impersonating a slave and attempting to perform a zone transfer from a master is a form of attack that attempts to map or footprint the zone. That is, by identifying host to IP address mappings, as well as other resource records, the attacker attempts to identify targets for direct attacks. Attacking a particular host using its hostname as a clue ("payroll," for example) provides an easy target to attempt access or denial of service (DOS).
- *Server Configuration*. An attacker may attempt to gain access to the physical server running the DNS service. Requiring a login and password to access the server locally or remotely is highly recommended to defend against direct server access. Use of secure shell (SSH) is also recommended for remote access. When using an IPAM system, verify that IPAM-to-DNS server communications are secure. Beyond being able to manipulate named and zone information, an attack of this type certainly enables the use of the server as a stepping stone to other targets, especially if this server happens to be trusted internally.
- *Control Channel Attack*. Access to ndc or rndc channels provides powerful remote control capabilities, such as stopping/halting named, reloading a zone, and more. Accessing the control channel and stopping the service thereby denies the service to querying servers and resolvers.
- *Buffer Overflows and Operating System Attacks*. An attacker may attempt to gain access to the server by overflowing the code execution stack or buffer. Without going into details, such an attack calls a subroutine that returns to the main program at a point defined by the attacker. This is one example of several similar types of OS level attacks, exploiting OS vulnerabilities on which the DNS service is running.
- *Configuration Errors*. While typically not malicious (though most attacks are initiated from internal sources), misconfiguring the DNS service and/or zone information may lead to improper resolution or server behavior.

12.1.3 Denial of Service Attacks

- *Denial of Service*. DNS, like other network services, is vulnerable to denial of service attacks, which features an attacker sending thousands of packets to a server in hopes of overloading the server, causing it to crash or become otherwise unavailable to other queriers. The service is rendered unavailable and thus denied to others.
- *Distributed Denial of Service*. A variant of this type of attack is the use of multiple distributed attack points and is referred to as a distributed denial of service (DDOS) attack. The intent is the same, though the scale is larger, potentially impacting several servers.
- *Reflector Attack*. This form of attack attempts to use DNS servers to launch massive amounts of data at a particular target, thereby denying service for the target machine. The attacker issues numerous queries to one or more DNS servers using the target machine's IP address as the source IP address in each DNS query. Querying for records with large quantities of data such as NAPTR, EDNS0, and DNSSEC queries magnifies this attack. Each responding server responds with the data to the "requestor" at the spoofed IP address to inundate this target with a large data flow.

12.2 MITIGATION APPROACHES

Strategies for addressing these vulnerabilities are summarized in the following table. In general, you should keep tabs on vulnerability reports from vendors and perform fixes and upgrades when they become available. Deployment strategies for hidden masters and generally deploying role-based DNS servers are also effective in mitigating attacks as discussed in Chapter 11. We'll discuss DNSSEC in detail in the next chapter.

Vulnerability	Mitigation
Packet interception/ spoofing	DNSSEC provides effective mitigation of this vulnerability by providing: • Origin authentication: verification of the data source • data integrity verification - data received is the same as the data published in the zone file • authenticated denial of existence - a sought resource record does not exist
ID guessing/query prediction	DNSSEC effectively mitigates this vulnerability; in addition, BIND 9 randomizes DNS header message IDs to reduce the chance of guessing its value in a fake response. Since mid July, 2008, BIND also randomizes UDP port numbers on outbound queries to reduce the risk of this vulnerability.

(*continued*)

(*Continued*)

Vulnerability	Mitigation
Name chaining/cache poisoning	DNSSEC provides origin authentication and data integrity verification to resist these vulnerabilities; Additional BIND directives for cache and additional section cache enabling and cleaning intervals can also help; transaction ID and UDP port randomization also help reduce the risk of this vulnerability
Resolver configuration attack	Configure DNS servers via DHCP; monitor or periodically audit clients for misconfigurations or anomalies
Illicit dynamic update	Use ACLs on allow-update, allow-notify, notify-source. ACLs may also be defined as requiring transaction signatures for added origin authentication
Illicit zone transfer	Use ACLs with TSIG on allow-transfer; and use transfer-source IP address and port to use a nonstandard port for zone transfers
Server attack/hijack	• Use hidden masters to inhibit detection of the zone master • Disallow recursive queries on masters and on ALL external DNS servers • Keep server operating system up to date • Limit port or console access • Implement chroot
Control channel attack	Use ACLs within the controls statement to restrict who can perform rndc commands; require rndc key
Buffer overflows and OS level attacks	Keep OS updated, limit cache, acache sizes and define cached cleaning intervals
Named service misconfiguration	Use checkzone and checkconf utilities, as well as an IPAM system with error checking; keep fresh backups for reload if needed
Denial of service	• Limit communications using rate limiting and such parameters as recursive-clients, max-clients-per-query, transfers-in, transfers-per-ns, cache and acache sizes; • Consider anycast deployment
Reflector attacks	• Use allow-query/allow-recursion ACLs • Use views if appropriate • Require TSIG on queries if possible

12.3 NON-DNSSEC SECURITY RECORDS

We'll cover DNSSEC in the next chapter but we conclude our pre-DNSSEC security chapter with a discussion of other security-oriented resource record types.

12.3.1 TSIG—Transaction Signature Record

Transaction signature (TSIG), defined in RFC 2845 (102), utilizes shared secret keys to establish a trust relationship between two DNS entities, whether two servers or a client

and a server. TSIG provides endpoint authentication and data integrity checking and can be used to sign dynamic updates and zone transfers. TSIG keys must be kept secure and manually configured on each end of the communications.

TSIG keys are used to sign a transaction by including a meta-resource record of type "TSIG" within the Additional section of a DNS message. A meta-resource record, similar to the OPT resource record type used for EDNS0, is used to pass additional information during a query/resolution transaction and is not included in a zone file per se. As such, these resource records are not cached and are computed dynamically for messages requiring signature.

The format of the TSIG meta-resource record is as follows:

Owner	TTL	Class	Type	RData							
Key Name	TTL	ANY	TSIG	Alg. Name	Time Signed	Fudge	MAC Size	MAC	Orig ID	Error	Other Len.	Other Data
k1-k2 ipamww.com.	0	ANY	TSIG	HMAC-MD5.SIG-ALG.REG.INT	232903 32	600	32	p19...	5076	0	0	

The RData fields within the TSIG meta record are defined as follows:

- *Algorithm Name*. The name of the hashing algorithm in domain name format. Currently defined algorithms are defined by IANA as follows:
  - HMAC-MD5.SIG-ALG.REG.INT (HMAC-MD5)
  - GSS-TSIG
  - HMAC-SHA1
  - HMAC-SHA224
  - HMAC-SHA256
  - HMAC-SHA384
  - HMAC-SHA512
- *Time Signed*. The time of signature in seconds since January 1, 1970 UTC.
- *Fudge*. Number of seconds of drift permitted in the Time Signed field.
- *MAC Size*. Length of the MAC in octets.
- *MAC*. The Message Authentication Code that contains a hash of the message being signed.
- *Original ID*. The ID number of the original message. If an update if forwarded, the message ID in the forwarded message could differ from the original. This enables the recipient to utilize the original message ID in reconstructing the original message for signature validation.
- *Error*. Encodes TSIG-related errors (see Table 9.1).
  - BADSIG: invalid key

- BADKEY: unknown key
- BADTIME: time signed outside of fudge range

- *Other Length*. Length in bytes of the Other Data section.
- *Other Data*. Blank unless Error = BADTIME, where the server will include its current time in this field.

The TSIG meta-resource record is constructed based on the message to be signed. A digest is created by applying the specified hash algorithm to the message and using this output as the Message Authentication Code field of the TSIG resource record. The TSIG meta-resource record is added to the Additional section of the DNS message.

12.3.2 SIG(0)—Signature Record with Empty Type Covered

Another form of transaction signature utilizes a special case of the SIG resource record, which was devised as part of the initial incarnation of DNSSEC. It has since been replaced by the RRSIG resource record in DNSSE*Cbis*. Nevertheless, a special case of the SIG resource record may be used independently of DNSSEC to sign updates and zone transfers. The format of the SIG resource record is shown below.

The notation SIG(0) refers to the use of the SIG resource record with an empty (i.e., 0) Type Covered field. In addition, RFC 2931 (118) recommends setting the owner field to root, the TTL to 0, and class to ANY as shown in the example below.

Owner	TTL	Class	Type	RData								
RRSet Domain	TTL	ANY	SIG	Type Cov.	Alg	Labels	Orig. TTL	Expire	Inception	Key tag	Signer	Signature
	0	ANY	SIG	0	3	3	86400	20080515 133509	2008011 5133509	30038	ipamww. com.	q8o1...

12.3.3 KEY—Key Record

The KEY record was defined with the initial definition of DNSSEC, but was superseded by the DNSKEY resource record. However, in the meantime, the KEY record was also utilized within SIG(0) to identify public keys used to decode signatures within the SIG(0) record. The KEY record has the same format as the DNSKEY record.

Owner	TTL	Class	Type	RData			
Key name	TTL	IN	KEY	Flags	Protocol	Algorithm	Key
K3941.ipamww.com.	86400	IN	KEY	256	3	1	12S9X-weE8F(le...

12.3.4 TKEY—Transaction Key Record

While RFC 2845 (102) specifies the TSIG standard, which utilizes shared secret keys, it does not provide for a key distribution or maintenance function. The Transaction Key (TKEY) meta-resource record was developed in order to support this key maintenance functionality. This process starts with a client or server sending a signed* TKEY query including any corresponding KEY records. A successful response from a server will include a TKEY resource record including an appropriate key. Depending on the mode specified in the TKEY record, both parties may now determine the shared secret. For example, if the Diffie–Hellman mode is specified, Diffie–Hellman keys are exchanged and both parties derive the shared secret that is then used to sign messages with TSIG.

The format of the TKEY meta-resource record is as follows:

Owner	TTL	Class	Type	RData								
Key Name	TTL	ANY	TKEY	Alg. Name	Inception	Expiration	Mode	Err	Key Size	Key Data	Other Len.	Other Data
k1-k2.ipamww.com.	0	ANY	TKEY	HMAC-MD5.SIG-ALG.REG.INT	232903 32	23300 6564	2	0	2048	9k)2 . . .	0	

The RData fields within the TKEY meta record are defined as follows:

- *Algorithm Name*. The name of the hashing algorithm in domain name format. Currently defined algorithms are defined by IANA as follows:
  - HMAC-MD5.SIG-ALG.REG.INT (HMAC-MD5)
  - GSS-TSIG
  - HMAC-SHA1
  - HMAC-SHA224
  - HMAC-SHA256
  - HMAC-SHA384
  - HMAC-SHA512
- *Inception*. The time of inception or beginning of validity of the key in seconds since January 1, 1970 UTC.
- *Expiration*. The time of expiration or ending of validity of the key in seconds since January 1, 1970 UTC.
- *Mode*. The form or scheme of key assignment, which may have the following values:
  - 0 = Reserved
  - 1 = Server assignment

* Yes, that's signed as in TSIG or SIG(0), so the initial condition requires a key, though TKEY provides a means to delete or update keys.

- 2 = Diffie–Hellman exchange
- 3 = GSS-API negotiation
- 4 = Resolver assignment
- 5 = Key deletion
- 6–65,534 = Available
- 65,535 = Reserved

- *Error*. Encodes TKEY-related errors (see Table 9.1)
  - BADSIG: invalid key
  - BADKEY: unknown key
  - BADTIME: time signed outside of inception/expiration range
  - BADMODE: specified mode not supported
  - BADNAME: invalid key name
  - BADALG: specified algorithm not supported
- *Key Size*. Size of the key data field in octets.
- *Key Data*. The key.
- *Other Length*. Not used.
- *Other Data*. Not used.