             Threat Analysis of the Domain Name System (DNS)

Status of this Memo

Copyright Notice

Abstract

   Although the DNS Security Extensions (DNSSEC) have been under
   development for most of the last decade, the IETF has never written
   down the specific set of threats against which DNSSEC is designed to
   protect. Among other drawbacks, this cart-before-the-horse situation
   has made it difficult to determine whether DNSSEC meets its design
   goals, since its design goals are not well specified. This note
   attempts to document some of the known threats to the DNS, and, in
   doing so, attempts to measure to what extent (if any) DNSSEC is a
   useful tool in defending against these threats.

1. Introduction

   The earliest organized work on DNSSEC within the IETF was an open
   design team meeting organized by members of the DNS working group in
   November 1993 at the 28th IETF meeting in Houston. The broad
   outlines of DNSSEC as we know it today are already clear in Jim
   Galvin's summary of the results of that meeting [Galvin93]:

   - While some participants in the meeting were interested in
     protecting against disclosure of DNS data to unauthorized parties,
     the design team made an explicit decision that "DNS data is
     'public'", and ruled all threats of data disclosure explicitly out
     of scope for DNSSEC.

   - While some participants in the meeting were interested in
     authentication of DNS clients and servers as a basis for access
     control, this work was also ruled out of scope for DNSSEC per se.

    - Backwards compatibility and co-existence with "insecure DNS" was
      listed as an explicit requirement.

    - The resulting list of desired security services was
      1) data integrity, and
      2) data origin authentication.

    - The design team noted that a digital signature mechanism would
      support the desired services.

While a number of detail decisions were yet to be made (and in some
cases remade after implementation experience) over the subsequent
decade, the basic model and design goals have remained fixed.

Nowhere, however, does any of the DNSSEC work attempt to specify in
any detail the sorts of attacks against which DNSSEC is intended to
protect, or the reasons behind the list of desired security services
that came out of the Houston meeting. For that, we have to go back
to a paper originally written by Steve Bellovin in 1990 but not
published until 1995, for reasons that Bellovin explained in the
paper's epilogue [Bellovin95].

While it may seem a bit strange to publish the threat analysis a
decade after starting work on the protocol designed to defend against
it, that is, nevertheless, what this note attempts to do. Better
late than never.

This note assumes that the reader is familiar with both the DNS and
with DNSSEC, and does not attempt to provide a tutorial on either.
The DNS documents most relevant to the subject of this note are:
[RFC1034], [RFC1035], section 6.1 of [RFC1123], [RFC2181], [RFC2308],
[RFC2671], [RFC2845], [RFC2930], [RFC3007], and [RFC2535].

For purposes of discussion, this note uses the term "DNSSEC" to refer
to the core hierarchical public key and signature mechanism specified
in the DNSSEC documents, and refers to TKEY and TSIG as separate
mechanisms, even though channel security mechanisms such as TKEY and
TSIG are also part of the larger problem of "securing DNS" and thus
are often considered part of the overall set of "DNS security
extensions". This is an arbitrary distinction that in part reflects
the way in which the protocol has evolved (introduction of a
putatively simpler channel security model for certain operations such
as zone transfers and dynamic update requests), and perhaps should be
changed in a future revision of this note.

2. Known Threats

   There are several distinct classes of threats to the DNS, most of
   which are DNS-related instances of more general problems, but a few
   of which are specific to peculiarities of the DNS protocol.

2.1. Packet Interception

   Some of the simplest threats against DNS are various forms of packet
   interception: monkey-in-the-middle attacks, eavesdropping on requests
   combined with spoofed responses that beat the real response back to
   the resolver, and so forth. In any of these scenarios, the attacker
   can simply tell either party (usually the resolver) whatever it wants
   that party to believe. While packet interception attacks are far
   from unique to DNS, DNS's usual behavior of sending an entire query
   or response in a single unsigned, unencrypted UDP packet makes these
   attacks particularly easy for any bad guy with the ability to
   intercept packets on a shared or transit network.

   To further complicate things, the DNS query the attacker intercepts
   may just be a means to an end for the attacker: the attacker might
   even choose to return the correct result in the answer section of a
   reply message while using other parts of the message to set the stage
   for something more complicated, for example, a name chaining attack
   (see section 2.3).

   While it certainly would be possible to sign DNS messages using a
   channel security mechanism such as TSIG or IPsec, or even to encrypt
   them using IPsec, this would not be a very good solution for
   interception attacks. First, this approach would impose a fairly
   high processing cost per DNS message, as well as a very high cost
   associated with establishing and maintaining bilateral trust
   relationships between all the parties that might be involved in
   resolving any particular query. For heavily used name servers (such
   as the servers for the root zone), this cost would almost certainly
   be prohibitively high. Even more important, however, is that the
   underlying trust model in such a design would be wrong, since at best
   it would only provide a hop-by-hop integrity check on DNS messages
   and would not provide any sort of end-to-end integrity check between
   the producer of DNS data (the zone administrator) and the consumer of
   DNS data (the application that triggered the query).

   By contrast, DNSSEC (when used properly) does provide an end-to-end
   data integrity check, and is thus a much better solution for this
   class of problems during basic DNS lookup operations.

TSIG does have its place in corners of the DNS protocol where there's
a specific trust relationship between a particular client and a
particular server, such as zone transfer, dynamic update, or a
resolver (stub or otherwise) that is not going to check all the
DNSSEC signatures itself.

Note that DNSSEC does not provide any protection against modification
of the DNS message header, so any properly paranoid resolver must:

- Perform all of the DNSSEC signature checking on its own,

- Use TSIG (or some equivalent mechanism) to ensure the integrity of
  its communication with whatever name servers it chooses to trust,
  or

- Resign itself to the possibility of being attacked via packet
  interception (and via other techniques discussed below).

2.2. ID Guessing and Query Prediction

Since DNS is for the most part used over UDP/IP, it is relatively
easy for an attacker to generate packets which will match the
transport protocol parameters. The ID field in the DNS header is
only a 16-bit field and the server UDP port associated with DNS is a
well-known value, so there are only 2**32 possible combinations of ID
and client UDP port for a given client and server. This is not a
particularly large range, and is not sufficient to protect against a
brute force search; furthermore, in practice both the client UDP port
and the ID can often be predicted from previous traffic, and it is
not uncommon for the client port to be a known fixed value as well
(due to firewalls or other restrictions), thus frequently reducing
the search space to a range smaller than 2**16.

By itself, ID guessing is not enough to allow an attacker to inject
bogus data, but combined with knowledge (or guesses) about QNAMEs and
QTYPEs for which a resolver might be querying, this leaves the
resolver only weakly defended against injection of bogus responses.

Since this attack relies on predicting a resolver's behavior, it's
most likely to be successful when the victim is in a known state,
whether because the victim rebooted recently, or because the victim's
behavior has been influenced by some other action by the attacker, or
because the victim is responding (in a predictable way) to some third
party action known to the attacker.

   This attack is both more and less difficult for the attacker than the
   simple interception attack described above: more difficult, because
   the attack only works when the attacker guesses correctly; less
   difficult, because the attacker doesn't need to be on a transit or
   shared network.

   In most other respects, this attack is similar to a packet
   interception attack. A resolver that checks DNSSEC signatures will
   be able to detect the forged response; resolvers that do not perform
   DNSSEC signature checking themselves should use TSIG or some
   equivalent mechanism to ensure the integrity of their communication
   with a recursive name server that does perform DNSSEC signature
   checking.

2.3. Name Chaining

   Perhaps the most interesting class of DNS-specific threats are the
   name chaining attacks. These are a subset of a larger class of
   name-based attacks, sometimes called "cache poisoning" attacks. Most
   name-based attacks can be partially mitigated by the long-standing
   defense of checking RRs in response messages for relevance to the
   original query, but such defenses do not catch name chaining attacks.
   There are several variations on the basic attack, but what they all
   have in common is that they all involve DNS RRs whose RDATA portion
   (right hand side) includes a DNS name (or, in a few cases, something
   that is not a DNS name but which directly maps to a DNS name). Any
   such RR is, at least in principle, a hook that lets an attacker feed
   bad data into a victim's cache, thus potentially subverting
   subsequent decisions based on DNS names.

   The worst examples in this class of RRs are CNAME, NS, and DNAME RRs
   because they can redirect a victim's query to a location of the
   attacker's choosing. RRs like MX and SRV are somewhat less
   dangerous, but in principle they can also be used to trigger further
   lookups at a location of the attacker's choosing. Address RR types
   such as A or AAAA don't have DNS names in their RDATA, but since the
   IN-ADDR.ARPA and IP6.ARPA trees are indexed using a DNS encoding of
   IPv4 and IPv6 addresses, these record types can also be used in a
   name chaining attack.

   The general form of a name chaining attack is something like this:

   - Victim issues a query, perhaps at the instigation of the attacker
     or some third party; in some cases the query itself may be
     unrelated to the name under attack (that is, the attacker is just
     using this query as a means to inject false information about some
     other name).

  - Attacker injects response, whether via packet interception, query
    guessing, or by being a legitimate name server that's involved at
    some point in the process of answering the query that the victim
    issued.

  - Attacker's response includes one or more RRs with DNS names in
    their RDATA; depending on which particular form this attack takes,
    the object may be to inject false data associated with those names
    into the victim's cache via the Additional section of this
    response, or may be to redirect the next stage of the query to a
    server of the attacker's choosing (in order to inject more complex
    lies into the victim's cache than will fit easily into a single
    response, or in order to place the lies in the Authority or Answer
    section of a response where they will have a better chance of
    sneaking past a resolver's defenses).

Any attacker who can insert resource records into a victim's cache
can almost certainly do some kind of damage, so there are cache
poisoning attacks which are not name chaining attacks in the sense
discussed here. However, in the case of name chaining attacks, the
cause and effect relationship between the initial attack and the
eventual result may be significantly more complex than in the other
forms of cache poisoning, so name chaining attacks merit special
attention.

The common thread in all of the name chaining attacks is that
response messages allow the attacker to introduce arbitrary DNS names
of the attacker's choosing and provide further information that the
attacker claims is associated with those names; unless the victim has
better knowledge of the data associated with those names, the victim
is going to have a hard time defending against this class of attacks.

This class of attack is particularly insidious given that it's quite
easy for an attacker to provoke a victim into querying for a
particular name of the attacker's choosing, for example, by embedding
a link to a 1x1-pixel "web bug" graphic in a piece of Text/HTML mail
to the victim. If the victim's mail reading program attempts to
follow such a link, the result will be a DNS query for a name chosen
by the attacker.

DNSSEC should provide a good defense against most (all?) variations
on this class of attack. By checking signatures, a resolver can
determine whether the data associated with a name really was inserted
by the delegated authority for that portion of the DNS name space.
More precisely, a resolver can determine whether the entity that
injected the data had access to an allegedly secret key whose

corresponding public key appears at an expected location in the DNS
name space with an expected chain of parental signatures that start
with a public key of which the resolver has prior knowledge.

DNSSEC signatures do not cover glue records, so there's still a
possibility of a name chaining attack involving glue, but with DNSSEC
it is possible to detect the attack by temporarily accepting the glue
in order to fetch the signed authoritative version of the same data,
then checking the signatures on the authoritative version.

2.4. Betrayal By Trusted Server

   Another variation on the packet interception attack is the trusted
   server that turns out not to be so trustworthy, whether by accident
   or by intent. Many client machines are only configured with stub
   resolvers, and use trusted servers to perform all of their DNS
   queries on their behalf. In many cases the trusted server is
   furnished by the user's ISP and advertised to the client via DHCP or
   PPP options. Besides accidental betrayal of this trust relationship
   (via server bugs, successful server break-ins, etc), the server
   itself may be configured to give back answers that are not what the
   user would expect, whether in an honest attempt to help the user or
   to promote some other goal such as furthering a business partnership
   between the ISP and some third party.

   This problem is particularly acute for frequent travelers who carry
   their own equipment and expect it to work in much the same way
   wherever they go. Such travelers need trustworthy DNS service
   without regard to who operates the network into which their equipment
   is currently plugged or what brand of middle boxes the local
   infrastructure might use.

   While the obvious solution to this problem would be for the client to
   choose a more trustworthy server, in practice this may not be an
   option for the client. In many network environments a client machine
   has only a limited set of recursive name servers from which to
   choose, and none of them may be particularly trustworthy. In extreme
   cases, port filtering or other forms of packet interception may
   prevent the client host from being able to run an iterative resolver
   even if the owner of the client machine is willing and able to do so.
   Thus, while the initial source of this problem is not a DNS protocol
   attack per se, this sort of betrayal is a threat to DNS clients, and
   simply switching to a different recursive name server is not an
   adequate defense.

   Viewed strictly from the DNS protocol standpoint, the only difference
   between this sort of betrayal and a packet interception attack is
   that in this case the client has voluntarily sent its request to the

attacker. The defense against this is the same as with a packet
interception attack: the resolver must either check DNSSEC signatures
itself or use TSIG (or equivalent) to authenticate the server that it
has chosen to trust. Note that use of TSIG does not by itself
guarantee that a name server is at all trustworthy: all TSIG can do
is help a resolver protect its communication with a name server that
it has already decided to trust for other reasons. Protecting a
resolver's communication with a server that's giving out bogus
answers is not particularly useful.

Also note that if the stub resolver does not trust the name server
that is doing work on its behalf and wants to check the DNSSEC
signatures itself, the resolver really does need to have independent
knowledge of the DNSSEC public key(s) it needs in order to perform
the check. Usually the public key for the root zone is enough, but
in some cases knowledge of additional keys may also be appropriate.

It is difficult to escape the conclusion that a properly paranoid
resolver must always perform its own signature checking, and that
this rule even applies to stub resolvers.

2.5. Denial of Service

As with any network service (or, indeed, almost any service of any
kind in any domain of discourse), DNS is vulnerable to denial of
service attacks. DNSSEC does not help this, and may in fact make the
problem worse for resolvers that check signatures, since checking
signatures both increases the processing cost per DNS message and in
some cases can also increase the number of messages needed to answer
a query. TSIG (and similar mechanisms) have equivalent problems.

DNS servers are also at risk of being used as denial of service
amplifiers, since DNS response packets tend to be significantly
longer than DNS query packets. Unsurprisingly, DNSSEC doesn't help
here either.

2.6. Authenticated Denial of Domain Names

Much discussion has taken place over the question of authenticated
denial of domain names. The particular question is whether there is
a requirement for authenticating the non-existence of a name. The
issue is whether the resolver should be able to detect when an
attacker removes RRs from a response.

General paranoia aside, the existence of RR types whose absence
causes an action other than immediate failure (such as missing MX and
SRV RRs, which fail over to A RRs) constitutes a real threat.
Arguably, in some cases, even the absence of an RR might be

considered a problem. The question remains: how serious is this
threat? Clearly the threat does exist; general paranoia says that
some day it'll be on the front page of some major newspaper, even if
we cannot conceive of a plausible scenario involving this attack
today. This implies that some mitigation of this risk is required.

Note that it's necessary to prove the non-existence of applicable
wildcard RRs as part of the authenticated denial mechanism, and that,
in a zone that is more than one label deep, such a proof may require
proving the non-existence of multiple discrete sets of wildcard RRs.

DNSSEC does include mechanisms which make it possible to determine
which authoritative names exist in a zone, and which authoritative
resource record types exist at those names. The DNSSEC protections
do not cover non-authoritative data such as glue records.

2.7. Wildcards

Much discussion has taken place over whether and how to provide data
integrity and data origin authentication for "wildcard" DNS names.
Conceptually, RRs with wildcard names are patterns for synthesizing
RRs on the fly according to the matching rules described in section
4.3.2 of RFC 1034. While the rules that control the behavior of
wildcard names have a few quirks that can make them a trap for the
unwary zone administrator, it's clear that a number of sites make
heavy use of wildcard RRs, particularly wildcard MX RRs.

In order to provide the desired services for wildcard RRs, we need to
do two things:

- We need a way to attest to the existence of the wildcard RR itself
  (that is, we need to show that the synthesis rule exists), and

- We need a way to attest to the non-existence of any RRs which, if
  they existed, would make the wildcard RR irrelevant according to
  the synthesis rules that govern the way in which wildcard RRs are
  used (that is, we need to show that the synthesis rule is
  applicable).

Note that this makes the wildcard mechanisms dependent upon the
authenticated denial mechanism described in the previous section.

DNSSEC includes mechanisms along the lines described above, which
make it possible for a resolver to verify that a name server applied
the wildcard expansion rules correctly when generating an answer.

3. Weaknesses of DNSSEC

   DNSSEC has some problems of its own:

   - DNSSEC is complex to implement and includes some nasty edge cases
     at the zone cuts that require very careful coding. Testbed
     experience to date suggests that trivial zone configuration errors
     or expired keys can cause serious problems for a DNSSEC-aware
     resolver, and that the current protocol's error reporting
     capabilities may leave something to be desired.

   - DNSSEC significantly increases the size of DNS response packets;
     among other issues, this makes DNSSEC-aware DNS servers even more
     effective as denial of service amplifiers.

   - DNSSEC answer validation increases the resolver's work load, since
     a DNSSEC-aware resolver will need to perform signature validation
     and in some cases will also need to issue further queries. This
     increased workload will also increase the time it takes to get an
     answer back to the original DNS client, which is likely to trigger
     both timeouts and re-queries in some cases. Arguably, many current
     DNS clients are already too impatient even before taking the
     further delays that DNSSEC will impose into account, but that topic
     is beyond the scope of this note.

   - Like DNS itself, DNSSEC's trust model is almost totally
     hierarchical. While DNSSEC does allow resolvers to have special
     additional knowledge of public keys beyond those for the root, in
     the general case the root key is the one that matters. Thus any
     compromise in any of the zones between the root and a particular
     target name can damage DNSSEC's ability to protect the integrity of
     data owned by that target name. This is not a change, since
     insecure DNS has the same model.

   - Key rollover at the root is really hard. Work to date has not even
     come close to adequately specifying how the root key rolls over, or
     even how it's configured in the first place.

   - DNSSEC creates a requirement of loose time synchronization between
     the validating resolver and the entity creating the DNSSEC
     signatures. Prior to DNSSEC, all time-related actions in DNS could
     be performed by a machine that only knew about "elapsed" or
     "relative" time. Because the validity period of a DNSSEC signature
     is based on "absolute" time, a validating resolver must have the
     same concept of absolute time as the zone signer in order to
     determine whether the signature is within its validity period or
     has expired. An attacker that can change a resolver's opinion of
     the current absolute time can fool the resolver using expired

signatures. An attacker that can change the zone signer's opinion
of the current absolute time can fool the zone signer into
generating signatures whose validity period does not match what the
signer intended.

- The possible existence of wildcard RRs in a zone complicates the
  authenticated denial mechanism considerably. For most of the
  decade that DNSSEC has been under development these issues were
  poorly understood. At various times there have been questions as
  to whether the authenticated denial mechanism is completely
  airtight and whether it would be worthwhile to optimize the
  authenticated denial mechanism for the common case in which
  wildcards are not present in a zone. However, the main problem is
  just the inherent complexity of the wildcard mechanism itself.
  This complexity probably makes the code for generating and checking
  authenticated denial attestations somewhat fragile, but since the
  alternative of giving up wildcards entirely is not practical due to
  widespread use, we are going to have to live with wildcards. The
  question just becomes one of whether or not the proposed
  optimizations would make DNSSEC's mechanisms more or less fragile.

- Even with DNSSEC, the class of attacks discussed in section 2.4 is
  not easy to defeat. In order for DNSSEC to be effective in this
  case, it must be possible to configure the resolver to expect
  certain categories of DNS records to be signed. This may require
  manual configuration of the resolver, especially during the initial
  DNSSEC rollout period when the resolver cannot reasonably expect
  the root and TLD zones to be signed.

4. Topics for Future Work

   This section lists a few subjects not covered above which probably
   need additional study, additional mechanisms, or both.

4.1. Interactions With Other Protocols

   The above discussion has concentrated exclusively on attacks within
   the boundaries of the DNS protocol itself, since those are (some of)
   the problems against which DNSSEC was intended to protect. There
   are, however, other potential problems at the boundaries where DNS
   interacts with other protocols.

4.2. Securing DNS Dynamic Update

   DNS dynamic update opens a number of potential problems when combined
   with DNSSEC. Dynamic update of a non-secure zone can use TSIG to
   authenticate the updating client to the server. While TSIG does not
   scale very well (it requires manual configuration of shared keys

between the DNS name server and each TSIG client), it works well in a
limited or closed environment such as a DHCP server updating a local
DNS name server.

Major issues arise when trying to use dynamic update on a secure
zone. TSIG can similarly be used in a limited fashion to
authenticate the client to the server, but TSIG only protects DNS
transactions, not the actual data, and the TSIG is not inserted into
the DNS zone, so resolvers cannot use the TSIG as a way of verifying
the changes to the zone. This means that either:

a) The updating client must have access to a zone-signing key in
   order to sign the update before sending it to the server, or

b) The DNS name server must have access to an online zone-signing key
   in order to sign the update.

In either case, a zone-signing key must be available to create signed
RRsets to place in the updated zone. The fact that this key must be
online (or at least available) is a potential security risk.

Dynamic update also requires an update to the SERIAL field of the
zone's SOA RR. In theory, this could also be handled via either of
the above options, but in practice (a) would almost certainly be
extremely fragile, so (b) is the only workable mechanism.

There are other threats in terms of describing the policy of who can
make what changes to which RRsets in the zone. The current access
control scheme in Secure Dynamic Update is fairly limited. There is
no way to give fine-grained access to updating DNS zone information
to multiple entities, each of whom may require different kinds of
access. For example, Alice may need to be able to add new nodes to
the zone or change existing nodes, but not remove them; Bob may need
to be able to remove zones but not add them; Carol may need to be
able to add, remove, or modify nodes, but only A records.

Scaling properties of the key management problem here are a
particular concern that needs more study.

4.3. Securing DNS Zone Replication

As discussed in previous sections, DNSSEC per se attempts to provide
data integrity and data origin authentication services on top of the
normal DNS query protocol. Using the terminology discussed in
[RFC3552], DNSSEC provides "object security" for the normal DNS query
protocol. For purposes of replicating entire DNS zones, however,
DNSSEC does not provide object security, because zones include
unsigned NS RRs and glue at delegation points. Use of TSIG to

protect zone transfer (AXFR or IXFR) operations provides "channel
security", but still does not provide object security for complete
zones. The trust relationships involved in zone transfer are still
very much a hop-by-hop matter of name server operators trusting other
name server operators rather than an end-to-end matter of name server
operators trusting zone administrators.

Zone object security was not an explicit design goal of DNSSEC, so
failure to provide this service should not be a surprise.
Nevertheless, there are some zone replication scenarios for which
this would be a very useful additional service, so this seems like a
useful area for future work. In theory it should not be difficult to
add zone object security as a backwards compatible enhancement to the
existing DNSSEC model, but the DNSEXT WG has not yet discussed either
the desirability of or the requirements for such an enhancement.

5. Conclusion

Based on the above analysis, the DNSSEC extensions do appear to solve
a set of problems that do need to be solved, and are worth deploying.

Security Considerations

This entire document is about security considerations of the DNS.
The authors believe that deploying DNSSEC will help to address some,
but not all, of the known threats to the DNS.

Acknowledgments

and any other members of the DNS, DNSSEC, DNSIND, and DNSEXT working
groups whose names and contributions the authors have forgotten, none
of whom are responsible for what the authors did with their ideas.

As with any work of this nature, the authors of this note acknowledge
that we are standing on the toes of those who have gone before us.
Readers interested in this subject may also wish to read
[Bellovin95], [Schuba93], and [Vixie95].

Normative References

   [RFC1034]   Mockapetris, P., "Domain names - concepts and
                 facilities", STD 13, RFC 1034, November 1987.

   [RFC1035]   Mockapetris, P., "Domain names - implementation and
                 specification", STD 13, RFC 1035, November 1987.

   [RFC1123]   Braden, R., "Requirements for Internet Hosts -
                 Application and Support", STD 3, RFC 1123, October 1989.

   [RFC2181]   Elz, R. and R. Bush, "Clarifications to the DNS
                 Specification", RFC 2181, July 1997.

   [RFC2308]   Andrews, M., "Negative Caching of DNS Queries (DNS
                 NCACHE)", RFC 2308, March 1998.

   [RFC2671]   Vixie, P., "Extension Mechanisms for DNS (EDNS0)", RFC
                 2671, August 1999.

   [RFC2845]   Vixie, P., Gudmundsson, O., Eastlake 3rd, D., and B.
                 Wellington, "Secret Key Transaction Authentication for
                 DNS (TSIG)", RFC 2845, May 2000.

   [RFC2930]   Eastlake 3rd, D., "Secret Key Establishment for DNS
                 (TKEY RR)", RFC 2930, September 2000.

   [RFC3007]   Wellington, B., "Secure Domain Name System (DNS) Dynamic
                 Update", RFC 3007, November 2000.

   [RFC2535]   Eastlake 3rd, D., "Domain Name System Security
                 Extensions", RFC 2535, March 1999.

Informative References

   [RFC3552]   Rescorla, E. and B. Korver, "Guidelines for Writing RFC
                 Text on Security Considerations", BCP 72, RFC 3552, July
                 2003.

   [Bellovin95] Bellovin, S., "Using the Domain Name System for System
                 Break-Ins", Proceedings of the Fifth Usenix Unix
                 Security Symposium, June 1995.

   [Galvin93]   Design team meeting summary message posted to dns-
                 security@tis.com mailing list by Jim Galvin on 19
                 November 1993.

   [Schuba93]   Schuba, C., "Addressing Weaknesses in the Domain Name
                 System Protocol", Master's thesis, Purdue University
                 Department of Computer Sciences, August 1993.

   [Vixie95]   Vixie, P, "DNS and BIND Security Issues", Proceedings of
                 the Fifth Usenix Unix Security Symposium, June 1995.

Authors' Addresses

   Derek Atkins
   IHTFP Consulting, Inc.
   6 Farragut Ave
   Somerville, MA 02144
   USA

   EMail: derek@ihtfp.com

   Rob Austein
   Internet Systems Consortium
   950 Charter Street
   Redwood City, CA 94063
   USA

   EMail: sra@isc.org

Full Copyright Statement

Intellectual Property

Acknowledgement