Using the Domain Name System for System Break-ins

Steven M. Bellovin
smb@research.att.com
AT&T Bell Laboratories

Abstract

The DARPA Internet uses the *Domain Name System* (DNS), a distributed database, to map host names to network addresses, and vice-versa. Using a vulnerability first noticed by P.V. Mockapetris, we demonstrate how the DNS can be abused to subvert system security. We also show what tools are useful to the attacker. Possible defenses against this attack, including one implemented by Berkeley in response to our reports of this problem, are discussed, and the limitations on their applicability are demonstrated.

This paper was written in 1990, and was withheld from publication by the author. The body of the paper is unchanged, even to the extreme of giving the size of the Internet as 200,000 hosts. An epilogue has been added that discusses why it was held back, and why it is now being released.

1 Introduction

In an earlier paper [Bel89], we discussed a number of security problems with the TCP/IP protocol suite. Many of them turned on the ability of an intruder to spoof the IP address of a trusted machine. In reality, though, hosts extend trust to other hosts based on their names, not their addresses; an attacker who can spoof a host's name can ignore the more difficult problem of faking its IP address. Some attacks along just these lines were mentioned in the earlier paper. In response to it, P.V. Mockapetris disclosed to us a more devastating attack based on the *Domain Name System* (DNS) [Moc87b, Moc87a]. Herein, we utilize his observations to invade selected machines, and demonstrate which other tools aided the attack.

Section 2 provides a brief overview of the DNS. Section 3 provides the details of some actual attacks. The names and IP addresses of the target hosts have been changed to protect the innocent. Section 4 discusses defenses, and shows why most of them are limited in their scope. We conclude by providing recommendations to software developers and system administrators.

Throughout this paper, we focus on the problem as it applies to Berkeley's remote login and remote shell services. We mention these specifically because they are ubiquitous, and the source code is readily available. Almost certainly, any other network services that rely on host names for authentication would be vulnerable. This probably includes various implementations of remote or networked file systems.

2 The Domain Name System

The DNS is a distributed data base used to map host names to IP (network layer) addresses, and vice-versa. The name space is divided into a series of *zones* based on syntactic separators (periods) in domain names. One or more servers contain the *authoritative* data for each zone. *Secondary* authoritative servers periodically poll the *primary* servers for their zones; if the data has changed, they initiate *zone transfer* operations to refresh their databases. At any level, a server may delegate the authority for a subdomain to a different server. Thus, if there is a server for a top-level domain `com`, it may itself contain the information for companies `small.com` and `smaller.com`, but delegate the responsibility for `monolith.com` to one or more of that company's machines. (In fact, servers for a domain need not, and often will not, reside within that domain.)

In general, hosts that use the DNS maintain local caches of the *resource records* returned. All resource records contain a Time-to-Live field set by the creator; at the end of that period, the cached record must be discarded, and an authoritative server queried anew.

Let us consider, as an example, the information for

Proceedings of the Fifth Usenix UNIX Security Symposium, June 1995, Salt Lake City, UT. ftp://ftp.research.att.com/dist/smb/dnshack.ps

```
$ORIGIN small.com
small.com.             IN      SOA     server.small.com. ghu.ws1.small.com. (
                                           901110001 ; Serial
                                           3600      ; Refresh
                                           600       ; Retry
                                           3600000   ; Expire
                                           86400 )   ; Minimum Time-to-Live
                       IN      NS      server
                       IN      NS      server.tiny.com.
server                 IN      A       222.33.44.1
                       IN      HINFO   Smallic/100 SmallIx
boss                   IN      A       222.33.44.2
                       IN      HINFO   Smallic/50 SmallIx
ws1                    IN      A       222.33.44.3
                       IN      HINFO   Smallic/40 SmallIx
ws2                    IN      A       222.33.44.4
                       IN      HINFO   Smallic/40 SmallIx

; Define a subdomain sales.small.com
sales                  IN      NS      thinker.sales.small.com.
                       IN      NS      ws1
droid.sales.small.com  IN      A       222.33.45.1
                       IN      A       222.33.44.5
```

Figure 1: The zone `small.com`.

zone `small.com`, as shown in Figure 1. The server contains a number of resource records.[1] Periods at the end of some names indicate that they are absolute, and not relative to the `$ORIGIN` field. An omitted name field from a record indicates that the name of the previous record should be used. And the ubiquitous `IN` field indicates that the records belong to the *Internet* domain; the DNS is capable of storing information about many different types of networks.

The `SOA` record defines the *Start of Authority* for the zone. Primarily a "glue record", it contains two fields of interest to us here: the machine that is the definitive source of the information in the zone, and the electronic mail address (in a variant form) of the person responsible. Note, incidentally, that the `SOA` record also contains the minimum expiration time for any resource records within the zone.

The `NS` records define the authoritative name servers for the domain `small.com`. Similar `NS` records must be in the server for the `com` domain so that inquiries may be directed to the proper place. In addition, the parent domain must contain `A` (address) records for all servers for its subdomains. This is illustrated by the records for `sales.small.com`, a subdomain under separate administration.

A host with more than one network interface will normally have multiple `A` records associated with it. Such hosts are often, but not always, gateways between the different networks.

Four hosts are defined within the domain, `server`, `boss`, `ws1`, and `ws2`. According to the `HINFO` record, all of the hosts appear to be Small, Inc.'s own computers and operating system.

A DNS query may request a record of a particular type—say, an `A` record—or it may ask for all records pertaining to a given name. A response may contain just the answers desired, a pointer to the proper server if the information is not contained within this zone, or an error indication if the record requested does not exist.

If appropriate, an answer may also contain *Additional Information* . Suppose a query were sent to the `small.com` server asking for the address of `critter.sales.small.com`. The reply would contain not just the `NS` record for `sales.small.com`, but also the `A` record for that server.

2.1 Inverse Mapping Domains

The records described above suffice for forward queries, where the client has a machine name and

[1] There are many other DNS records than those described here. We are mentioning only those that provide information necessary to understand the subsequent discussions.

wishes to look up the IP address. However, ordinary DBMS-style inverse queries, to map addresses into names, do not work.

The reason why is a bit subtle. Certainly, one could ask any given server which machine corresponded to the address `222.33.45.100`. Unfortunately, that server would be unlikely to know the answer. Nor, without more information, could the client know which server to query. The DNS is a distributed database, with boundaries delimited by host name syntax. IP addresses contain no clues to this organization.

Instead, inverse mappings are implemented by a separate, parallel tree, keyed by IP address. To mimic the structure of IP addresses, and hence the fashion in which they are assigned to administrators, the individual bytes of the address are reversed. A standard suffix is appended to avoid name collisions with forward mappings. Thus, the address-to-name mapping for host `ws1`, which has an IP address of `222.33.44.3`, would be handled by a server for zone `44.33.222.in-addr.arpa`. Its database, with glue records omitted, looks like this:

```
$ORIGIN 44.33.222.in-addr.arpa
1        IN      PTR      server.small.com.
2        IN      PTR      boss.small.com.
3        IN      PTR      ws1.small.com.
4        IN      PTR      ws2.small.com.
```

Each record contains only a pointer to the forward mapping record. Generally, the inverse mapping tree will reside on the same machine as the corresponding forward mapping tree, but this is not a requirement. Hosts with more than one address will have `PTR` records in more than one inverse mapping tree.

3 Attack!

> *Kids! Do not try this at home! This stunt was carried out by trained professionals, who—apart from knowing what they were doing—had the permission of the relevant system administrators!*

Our threat scenario assumes that the attacker has complete control of a machine containing legitimate primary servers for a DNS zone, including the associated inverse mapping tree. That is, he or she can make whatever changes are desired to the delegated portions of the name tree, and such changes will be accepted as correct by other machines, subject only to the usual expiration dates. We also assume the ability to control any TCP port numbers on that machine, including those that Berkeley's versions of the UNIX system regard as privileged. The attacker may be either a renegade system administrator or someone who has successfully subverted a DNS server machine. History demonstrates that both possibilities are real.

The attacker's goal is to find hosts that trust other hosts using their names, and to learn the names of some of those trusted partners. While random patterns of trust can and do exist, a more fruitful approach is to look for two common patterns. First, in a cluster of time-sharing machines, each of the machines is likely to extend blanket trust to the others. Even if that does not apply to the general user population, it probably does apply to the systems programming and operational staffs. Second, the attacker can look for file servers and their workstations. The file servers sometimes trust their clients, serving as a source of extra CPU cycles. Furthermore, if the clients are "dataless", they will frequently trust an administrative machine to permit software maintenance.

In the following examples, we will assume that the target organization has the following machines:

Name	*IP Address*	*Type*
`bullseye.softy.org`	192.193.194.1	file server
`ringer.softy.org`	192.193.194.64	workstation
`groundzero.softy.org`	192.193.194.65	workstation

All are running some derivative of 4.2BSD or 4.3BSD, such as SunOS, and all trust each other via `/etc/hosts.equiv` files.

The attacker, whom we shall dub *Cuckoo* in honor of Cliff Stoll's book [Sto89], is coming from machine `cracker.ritts.org`, `150.151.152.153`.

The essence of the basic attack relies on the nature of the address-to-name mapping. As noted above, this mapping uses an independent DNS tree. Assume that the inverse mapping record for `150.151.152.153` is changed from the correct `cracker.ritts.org` to `ringer.softy.org`. When the attacker attempts to `rlogin` to `bullseye`, it will try to validate the *name* —not the IP address— of the calling machine. It does this by calling `gethostbyaddr()` and passing it the address `150.151.152.153`. In a DNS environment, that call translates to a name server query for the record associated with `153.152.151.150.in-addr.arpa`. This will, of course, retrieve the `PTR` record shown above. Thus, `bullseye` believes that its friend and neighbor `ringer` is trying to connect. Thus, the call is accepted, and the attack has succeeded.

Note the fundamental flaw we have exploited here. There is no forced linkage between the two DNS trees owned by Cuckoo, `ritts.org` and 152.151.150.in-addr.arpa, even though they should contain complementary data. Thus, the latter tree can contain entries pointing to hosts belonging to Softies, Inc.

3.1 Filling in the Blanks

In order to mount the attack described above, Cuckoo needs to know three things: a target host name, a user name to impersonate, and a machine trusted by the target host. There are many approaches possible, all using a variety of standard tools. The following sequence, which we actually used in our first demonstration attack, is illustrative.[2] We will start by assuming we know the name of the target machine, perhaps from a mail message or news article. Next, we use SNMP [CDF89] to examine its TCP connection tables (Figure 2). That is, apart from the miscellaneous connection to port 411 on some other host, the other TCP activity represents uses of `rlogin`. One connection is from `ringer`; the others represent the two endpoints of an `rlogin` session from `bullseye` to itself. This is an odd circumstance, and potentially quite revealing. The `finger` command tells us more (Figure 3).

Two entries indicate remote logins. On `ttyp4`, user `random` is connected from `ringer`. And on `ttyp5`, `bingo` is connected from `bullseye` itself, in a loopback connection. It seems likely that this represents `user1` utilizing a different login, rather than `random` doing so; such a connection from `random` would more likely have originated from `ringer`. So we have a possible `.rhosts` file for `bingo` authorizing a local user `user1` when coming from host `bullseye`. We can also try for `random` coming from `ringer`; a quick check with `finger` shows a user of the same name logged in there.

The remaining steps are trivial. We modify the `PTR` record for `cracker` appropriately, create local login names `user1` and `random` for ourselves, and use them to attempt an `rlogin` to the target machine. The attack succeeded on the second try; it turned out that `random's` connection was not preauthorized.

Actually, the procedure outlined above got a bit tedious, so we developed some extra tools to help. First, we installed several instances of the pseudonetwork driver [Bel90], not because we needed its facilities but because it gave us an easy way to associate several new IP address with our machine without installing any extra hardware. This let us attempt impersonations of several machines at once.

[2] The output you are about to see is real. Only the names and IP addresses have been changed, to protect the innocent.

Slight modifications to the global routing tables, courtesy of `routed`, were needed to route packets back to us. We also modified `rlogin` to let us specify the local host address and the local user name, thus avoiding the need for extra `/etc/passwd` entries. Our attacks became this simple:

```
xrlogin bullseye.softy.org -l bingo \
   -L user1 -x 150.151.252.153
```

The appropriate `PTR` records were all created together, of course.

3.2 Other Approaches

Some might object that the penetration described above relied on SNMP, a facility not often found on hosts, and on `finger`, a service which sometimes does not give the name of the remote hosts. Fortunately for the attacker, there are other ways to gather the necessary data.

One useful tool is electronic mail. Suppose you wish to target someone who sent you (or a mailing list) some electronic mail. These days, mail headers are often modified to indicate that the sender's machine is some gateway, perhaps a file server. But the `Received:` or `Message-Id:` lines indicate the actual source of the mail. Often, that user will be able to `rlogin` to the apparent sending machine—i.e., the file server—from the actual sending machine, typically a workstation.

Another useful tool is the DNS itself; it contains a wealth of information. The `SOA` record contains the address of a privileged user and a machine containing data administered by that person. That alone is a useful pair to attack. If that fails, assume again that we know a user name and a machine name. The DNS will tell us the IP address of that machine. Generally, it will also permit a zone transfer to list the other machines on that network. Thus, if we only know one machine name, `groundzero.softy.org`, but do not know any other machines in that organization, we can quickly learn that `groundzero` is `192.193.194.65`. Next, we try a zone transfer request for `194.193.192.in-addr.arpa`, using any of a number of standard tools. (If the zone transfer is rejected, we only need to issue 254 individual queries to map a Class C network.) That gives us the names of other machines on the same network. If we're lucky, the DNS entries for those machines will include `HINFO` lines; the model numbers provide powerful clues as to which machines are file servers and which are workstations. (SNMP, if available on the targets, can also provide model numbers.) Further clues can be sometimes be found by use of

```
$ snmpnetstat bullseye.softy.org public
Active Internet Connections
Proto Recv-Q Send-Q Local Address              Foreign Address            (state)
tcp      0      0   bullseye.softy.org.login  bullseye.softy.org.1023    ESTAB
tcp      0      0   bullseye.softy.org.login  ringer.softy.org.1020      ESTAB
tcp      0      0   bullseye.softy.org.1023   bullseye.softy.org.login   ESTAB
tcp      0      0   bullseye.softy.org.3593   other.host.com.411         ESTAB
```

Figure 2: Connection patterns via SNMP.

`finger` (which machines have multiple users logged in?), `SMTP` (does the machine run a mail server?), anonymous `FTP` (workstations rarely offer the service; servers sometimes do), and Sun's `rpcinfo` (what services are running?). It may not even matter very much—some organizations use the same `/etc/hosts.equiv` file on all of their machines, just to simplify system administration.

4 Attempted Defenses

A variety of defenses have been tried or contemplated; few are generally successful. The first was developed by Berkeley when we reported this problem some time ago. It consists of modifications to `rlogind` and `rshd` to validate the inverse-mapping tree by looking at the corresponding node on the forward-mapping tree. That is, if the `gethostbyaddr`() call on address `150.151.152.153` returns the name `bullseye.softy.org`, the server will issue a `gethostbyname`() call on that name, and the list of addresses returned is matched against `150.151.152.153`. If the match fails, a impersonation attempt is flagged. In the general case, this defense is easily countered. To see how, let us analyze the transactions in terms of the DNS.

The `gethostbyaddr`() call is, as noted, implemented by a DNS request for a `PTR` record. The server that supplies this PTR record is under Cuckoo's control, and may return false information. The `gethostbyname`() call requests `A` records ultimately from the server for `softy.org`, which is not controlled by the attacker. However, in reality the query does not go immediately to that server; rather, it goes to the local machine's name server. And that server has a cache which may be poisoned by Cuckoo. Specifically, the DNS message containing the `PTR` record may contain a bogus `A` record in its Additional Information field. The information in this record will be returned on any `gethostbyname`() call, along with other (presumably legitimate) `A` records. The name server for 4.3BSD does not normally include any `A` records when sending out `PTR` responses, but the modification to make it do so is trivial. The results of the change are shown in Figure 4, using Mockapetris's `dig` program.

Note the very short time-to-live fields; the attacker does not want these anomalous records staying around where they might be observed. This is particularly necessary for the `A` record; it might be embarrassing if it were returned to a legitimate user seeking the address of `bullseye.`

It has been suggested to us that additional information fields be scrutinized more carefully before acceptance. In the above example, there is little reason to include an `A` record with a `PTR`; would it help if the name server rejected it? Again, the answer is no; there are other, apparently legitimate, ways to introduce bogus `A` records. For example, one can often persuade a host to do a lookup for a hostname in a subdomain of `ritts.org`, say `foo.bar.ritts.org`. Such a query will (properly) be answered by the server for domain `ritts.org`, which is controlled by the attacker. The response will contain a list of the name servers for domain `bar.ritts.org`. Those `NS` records are, of necessity, accompanied by the corresponding `A` records. Nothing would prevent the inclusion of `bullseye.softy.org` and the corresponding fraudulent `A` record in this list.

Attempts to poison the cache of a primary or secondary server for a domain do not work. The standard name servers will reject updates to zones for which they are authoritative. Thus, the attacker cannot insert bogus `A` records, so the cross-check will detect the attack. This prohibition is only reasonable, and not just for security reasons; an authoritative server by definition possesses all possible data for its zone. Attempts to update the zone remotely are at best naive.

In some environments, this provides a reasonably strong defense. Most `rlogin` and `rsh` requests do come from the local cluster of machines, and it is precisely for these that the local server is authoritative. Nevertheless, care should be taken. Caching-only servers are *not* immune, as they possess no authoritative data of their own. Nor are authorita-

```
$ finger @bullseye.softy.org
[bullseye.softy.org]
Login    Name           TTY   Idle   When   Where
user1    User One       co           Fri    13:18
user1    User One       p0    1:48   Mon    13:15   unix:0.0
user1    User One       p1      3d   Mon    13:15   unix:0.0
user1    User One       p2           Mon    13:15   unix:0.0
user1    User One       p3    1:56   Wed    12:45   unix:0.0
random   Amber Random   p4      3d   Wed    15:51   ringer.softy.org
bingo    Bingo Scores   p5    1:56   Wed    12:46   bullseye.softy.org
user1    User One       p6      12   Fri    12:15   unix:0.0
```

Figure 3: Learning from the `finger` command.

tive servers when fielding requests from outside their zone; if a host trusts another host not named in a local zone, its name server cannot protect it.

The target is also in a somewhat stronger position if it is a secondary server for the inverse mapping domain of the attacker. In that case, the `PTR` record will be retrieved from an unmodified name daemon; hence, the attacker will not be able to add the Additional Information field. This, too, can be countered. In a variant analagous to one discussed earlier, the attacker can create an `NS` record for the inverse domain naming the impersonated machine as a secondary server; in such cases, the fraudulent `A` record will be sent along on zone transfer requests.

Another layer of protection can be provided if the target host uses a local mapping table before consulting the DNS. For example, some sites use Sun's interface between Network Information Service (NIS, nee YP) and the DNS. On such machines, the DNS is queried only if NIS does not have the answer. Thus, the inverse lookup will retrieve the bogus `PTR` record, since the address used is not in the local host tables. But the forward lookup—the `gethostbyname`() call—will be satisfied by NIS without resort to the DNS, and hence without retrieving the poisoned `A` record.

It must be underscored that this defense[3] does not work at all if the attacker finds a target user coming in from a host not listed in the NIS database. Conceptually, it is analogous to the situation of authoritative DNS servers: the target host possesses incorruptible[4] local information.

5 Hardening DNS Servers

It would be very useful, when tracking these and other problems, if DNS server cache entries were tagged with their source. Thus, bogus `A` records could be tracked back. Note that this is not just a security issue; periodically, assorted newsgroups and mailing lists discuss why a DNS zone has been corrupted, and how to purge the offending entries. The ability to trace the offending records to their source would help tremendously.

Preventing cache contamination is probably not feasible. If a server is not authoritative for a zone, it has no way of knowing whether or not Additional Information records may be trusted. It does no good to add authentication to DNS responses; the attacks described herein all come from the legitimate (albeit untrustworthy) sources. In some implementations, it might be feasible to restrict the scope of the cache. Perhaps Additional Information records should be used only when resolving particular queries, and then discarded. That is harder to do for `NS` records, but the associated `A` records could be bound to the name server definitions and not used for other purposes. Obviously, any tinkering along these lines would result in a smaller, less useful, cache. And that in turn would lead to more DNS queries, possibly an unacceptable price.

6 Conclusions and Recommendations

As we have stated before [Bel89], reliance on host addresses or host names for authentication is fundamentally flawed. The only real security in an internetworking environment is cryptographic. The Kerberos system [Bry88, KN93, MNSS87, SNS88, NT94]

[3] In SunOS 4.1 and later, the cross-check is implemented in the `gethostbyaddr`() subroutine; thus, all utilities reap the benefits of increased security. However, the message generated in case of a failure does not indicate a possible security problem. Judging from comments on assorted mailing lists, this possibility is not well known. Furthermore, there are reports that a bug in some versions causes frequent erroneous generation of this message, thus lulling even alert administrators.

[4] Security holes in NIS are beyond the scope of this paper.

```
$ dig -x 150.151.152.153 @server.ritts.org

; <<>> DiG 2.0 <<>> -x @server.ritts.org
;; ->>HEADER<<- opcode: QUERY , status: NOERROR, id: 10
;; flags: qr aa rd ra ; Ques: 1, Ans: 1, Auth: 0, Addit: 2
;; QUESTIONS:
;;      153.252.151.150.in-addr.arpa, type = ANY, class = IN

;; ANSWERS:
153.252.151.150.in-addr.arpa.   30      PTR     bullseye.softy.org.

;; ADDITIONAL RECORDS:
bullseye.softy.org.             15      A       150.151.252.153

;; Sent 1 pkts, answer found in time: 70 msec
;; FROM: cracker to SERVER: server.ritts.org  150.151.152.154
;; WHEN: Tue Oct 30 13:20:54 1990
```

Figure 4: Returning an addtional `A` record when a `PTR` record has been requested.

is probably the best choice today; though flawed in places [BM91], it is far better than the current scheme.

If it is not possible to use cryptographic authentication, installation of Berkeley's fix is mandatory. Without it, none of the palliative strategies described below do any good.

A first cut at protection, in such cases, would be to restrict name-based authentication to hosts blessed, as it were, by the system administrator. Though there are other obvious security benefits, the advantage here is that the set of trusted hosts would be limited to those for which the local machine has authoritative name information. Berkeley's latest versions of `rlogind` and `rshd` support this by permitting the administrator to disable use of `.rhosts` files.

If this is not feasible, an alternative is to have the local name server act as a secondary server for important neighboring zones, and thus possess authoritative forward-mapping data. In many environments, most remote login requests will come from a very few other organizations; one need not download all mapping information for the entire network. Furthermore, with current implementations, such secondary servers need not be "official". That is, if `depta.company.com` wished to be a secondary server for the `deptb.company.com` domain, it is not necessary for the administrator of the `company.com` zone to create any new `NS` records. In most cases, the zone transfer will succeed, and `depta`'s domain server will possess correct data.[5]

A corollary to this is that caching-only servers are Bad. They possess no authoritative data of any sort, and are very susceptible to cache poisoning. *If DNS attacks are possible, no host should rely on any caching-only server for information.* All time-sharing machines within an organization, and all file servers, should possess definitive mapping information for the hosts within the organization. This may be accomplished via NIS, DNS secondary servers, or other means. This in turn implies that huge domains—those encompassing entire campuses, for example—are probably a bad idea, as far too many machines would have to possess far too much data.

6.1 Logging and Auditing

As always, proper logging and auditing can be significant aids in detecting DNS attacks. For example, the anti-spoofing code added to `rlogind` and `rshd` should inform the system administrator of such attempts. The versions released by Berkeley inform the attacker alone. Similarly, it would be useful if DNS servers logged attempts to update authoritative zones. This is not a trivial task, as there are a number of contexts in which it is legitimate to receive locally-known data in Additional Information records, but it might be feasible. As a rule of thumb, one should log any responses that not only refer to a local zone, but also either attempt to add new records, or to add contradictory information to existing records other than time-to-live. There will be noise in the log even so, especially if some records

[5] Do note, though, that some organizations block zone transfers from unapproved sites. In any event, politeness would seem to dictate that `deptb`'s administrator be contacted first.

have been changed but stale copies still exist elsewhere.

A second useful log would be of all connection attempts, successful or not, to `rlogind` or `rshd`. These should include the remote host name, the IP address (of course), and the local and remote user names. This file may be audited offline to check for mismatches between the host name and address. Obviously, such checks require great care, lest they be spoofed by the same attack.

A more difficult detection measure would involve comparisons of the forward-mapping data against the inverse mapping data for the zone. Presumably, a security organization could attempt random zone transfers from various authorized servers, and audit the data retrieved. It is unclear if such matches are actually useful. Apart from the obvious—a clever attacker will not leave bogus resource records around when not using them—inverse mapping domains are notorious for their poor quality. (To be sure, finding such errors might justify the effort, even apart from security considerations.) Additionally, unusual situations are often created deliberately to deal with multi-homed hosts, or to create pseudo-hosts for specific services. Detecting attacks amidst this sort of noise is quite hard.

6.2 Giving Away Information

The astute reader will have noticed that the attack scenarios we have presented relied on gathering trust data first. Put another way, we abused assorted standard services to learn which pairs of hosts were worth attacking. As we have noted earlier [Bel89], any sort of information utility— `finger`, SNMP, etc.—provides a wealth of information to an attacker. System administrators should ask themselves if the benefits of these utilities outweigh the risks.

6.3 Should the DNS be Abandoned?

In response to reports of these problems, some individuals have suggested that the DNS be abandoned, in favor of a return to static host tables. We do not agree with this suggestion, for several reasons.

First and foremost, the problem here is not the DNS, it is inadequate methods of host authentication. If strong (i.e., cryptographic) mechanisms were used, people playing games with inverse mapping records would be notable solely for their nuisance value. At most, some log files would need to be enhanced to record IP addresses as well as host names.

Nor would abandoning the DNS solve the problem of attacks that actually mimic the IP address of a trusted host, rather than simply its name. Such attacks are somewhat harder, but the techniques have been published for some time. (See, for example, [Mor85].) Trying to fix the name table problem instead of tackling the real issue is treating the symptoms, not curing a disease.

Second, it is by no means clear that the DNS is the real source of the problem. Rather, the problem arises because the *information* necessary to compile any sort of host-to-address mapping scheme is of necessity distributed in nature. The administrators involved are located around the world. None of the usual mechanisms for transmitting updates to a central site (electronic mail, telephone calls, or even conventional paper mail) carry any strong authentication; a table compiler who blithely acts on all update requests can install poisoned records almost as easily as can a DNS administrator.

Finally, abandoning the DNS would leave unsolved the problem it was originally meant to tackle: the sheer size of the host table. By best estimates the Internet is comprised of more than 200,000 machines; neither the update frequency nor the timely distribution of new tables can be handled by other mechanisms.

7 Acknowledgements

We wish to thank the anonymous system administrators who consented to have their system attacked. In addition, they graciously installed test domains, new versions of `rlogin` and `rshd`, and other files we requested to either facilitate or defeat new attempts at penetration.

8 Epilogue

As noted, this paper has been withheld by the author for over four years. Holding it back was not an easy decision, nor was eventual decision to release it. For both decisions, we cite the external world; as it has changed, so has the outcome.

The paper was held back—not suppressed; no external agency applied any pressure, though there were certainly others who were happy it was not published at the time—because it described a serious vulnerability for which there was no feasible fix. The only choice would have been to give up entirely on name-based authentication, a choice the industry was not able to make in 1990.

Attitudes and technology have changed since then. For one thing, many sites already use firewalls to *protect against such attack; if an attacker cannot open an* `rsh` *connection, he or she cannot claim a false origin for it. The recent successful sequence number guessing attack*[6] *has probably sealed the doom of name-based authentication over the Internet.*

Cryptography is also more accepted. Apart from its direct use for authenticating connections, there are now proposals for cryptographic authentication of the DNS itself [EK96]. That would be a complete defense against the cache poisoning attacks described here; DNS responses would be self-authenticating, and forged responses would be detected and dropped.

There has also been a proposal for how name servers can defend against cache contamination. The concept is simple: only use Additional Information resource records in the context in which they were returned. That is, an `A` *record that accompanied an* `MX` *record would be consulted only when sending mail to that site. It would not be used when a general address lookup was done, or to confirm the names received via* `PTR` *records. This and other enhancements are described further by Vixie [Vix95].*

Arguably, this paper should have been published when written. "Death of the Net predicted; film at 11" is an old refrain, and the Net has now survived password sniffers and sequence number attacks. More to the point, if more people had known of the attack, perhaps the solution described above would have been found sooner.

Finally, the secrecy may have been in vain. Apart from reports that this exact technique was used by hackers many years ago—and the reports are quite reliable—the paper leaked anyway. We have seen it on at least one Web server, and follow-up work by Schuba has been available for quite some time [SS93].

References

[Bel89] Steven M. Bellovin. Security problems in the TCP/IP protocol suite. *Computer Communications Review*, 19(2):32–48, April 1989.

[Bel90] Steven M. Bellovin. Pseudo-network drivers and virtual networks. In *USENIX Conference Proceedings*, pages 229–244, Washington, D.C., January 22-26, 1990.

[BM91] Steven M. Bellovin and Michael Merritt. Limitations of the Kerberos authentication system. In *USENIX Conference Proceedings*, pages 253–267, Dallas, TX, Winter 1991.

[Bry88] B. Bryant. Designing an authentication system: A dialogue in four scenes, February 8, 1988. Draft.

[CDF89] J. Case, C. Davin, and M. Fedor. Simple network management protocol SNMP. RFC 1098, Internet Engineering Task Force, April 1989. (Obsoletes RFC1067); (Updated by RFC1157).

[EK96] Donald E. Eastlake, 3rd and Charles W. Kaufman. Domain name system protocol security extensions. Internet draft; work in progress, January 30, 1996. (draft-ietf-dnssec-secext-09.txt).

[KN93] J. Kohl and B. Neuman. The kerberos network authentication service (V5). Request for Comments (Proposed Standard) RFC 1510, Internet Engineering Task Force, September 1993.

[MNSS87] S. P. Miller, B. C. Neuman, J. I. Schiller, and J. H. Saltzer. Kerberos authentication and authorization system. In *Project Athena Technical Plan*. MIT, December 1987. Section E.2.1.

[Moc87a] P. Mockapetris. Domain names - concepts and facilities. Request for Comments (Standard) STD 13, RFC 1034, Internet Engineering Task Force, November 1987. (Obsoletes RFC0973); (Updated by RFC1101).

[Moc87b] P. Mockapetris. Domain names - implementation and specification. Request for Comments (Standard) STD 13, RFC 1035, Internet Engineering Task Force, November 1987. (Obsoletes RFC0973); (Updated by RFC1348).

[Mor85] Robert T. Morris. A weakness in the 4.2BSD UNIX TCP/IP software. Computing Science Technical Report 117, AT&T Bell Laboratories, Murray Hill, NJ, February 1985.

[NT94] B. Clifford Neuman and Theodore Ts'o. Kerberos: An authentication service for computer networks. *IEEE Communications*, 32(9):33–38, September 1994.

[6] CERT Advisory CA-95:01, January 23, 1995

[SNS88] Jennifer Steiner, B. Clifford Neuman, and Jeffrey I. Schiller. Kerberos: An authentication service for open network systems. In *Proc. Winter USENIX Conference*, pages 191–202, Dallas, TX, 1988.

[SS93] Christoph L. Schuba and Eugene H. Spafford. Addressing weaknesses in the domain name system protocol. Master's thesis, Purdue University, 1993. Department of Computer Sciences.

[Sto89] Cliff Stoll. *The Cuckoo's Egg: Tracking a Spy Through the Maze of Computer Espionage*. Doubleday, New York, 1989.

[Vix95] Paul Vixie. DNS and BIND security issues. In *Proceedings of the Fifth Usenix* UNIX *Security Symposium*, pages 209–216, Salt Lake City, UT, 1995.