Countering Abuse of Name–Based Authentication [1]

Christoph L. Schuba and Eugene H. Spafford

COAST Laboratory
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907-1398
{schuba,spaf}@cs.purdue.edu

Abstract

Authentication for access control procedures is usually based on the identity of participating entities. In some communications systems, identities are partially or wholly resolved using hostnames or machine addresses in the underlying protocol suite. Access control lists and revocation lists are often defined on the basis of hostnames, whereby the communication subsystem at runtime utilizes machine addresses.

After communications between two machines are established, hosts identify each other by their protocol addresses. To map this address to a high–level name, which can then be compared with access control or revocation lists to grant or deny access, a resolution process is initiated. The abstraction from protocol addresses to high–level hostnames is necessary to hide details of heterogeneous communication subsystems, and of dynamic network configurations from the application layer where a uniform, high–level naming scheme is desired.

If cryptographic capabilities are used that identify subject–object interactions, authentication usually does not depend on host identification. Where host identification is part of the authentication, a crucial link in the chain of authentication is the association between hostnames and their respective addresses. The validity of the authentication can be trusted only as much as the binding process itself.

In the Internet this name resolution is provided by a widely–implemented distributed database system: the Domain Name System (DNS). Dynamic configuration behavior, system efficiency, and volume of binding requests demand late binding between hostnames and addresses, and caching of the mappings. Therefore, bindings are established "just in time" on a need basis and are kept valid for a limited period of time.

[1] submitted to the twenty-second annual *Telecommunications Policy Research Conference*

This paper describes problems of name–based authentication requiring late binding such as that provided by the DNS for hostname–to–address associations. Because forward mappings (where the address is a relation of the hostname) and reverse mappings are maintained in unrelated parts of the database, three levels of modification are possible: modified forward mapping, modified backward mapping, or both. The modification of associations enables the spoofing of hostnames in sessions that depend on the DNS.

We state the problem in an abstract way and in the concrete case of the DNS. We analyze the conditions that facilitate the exploitation of the problem and explain the weaknesses that are present in the DNS.

We then explore some possible solutions to the problem. All our proposed solutions are evaluated by a number of criteria to compare effects of the solutions. Each of the solutions will either consist of mechanisms that enable arbitrarily chosen policies, or it will require the implementation of a certain policy. We emphasize the solutions to improve existing name servers by modifying them in a way that they rely on less trust, and to embed crytographic methods into the name resolution process.

1 Introduction

The Internet is a widespread conglomeration of hundreds of thousands of interconnected heterogeneous networks and hosts. The design of the Internet is based on a protocol hierarchy. There exist multiple implementations of these protocols.

Computers communicate with each other on the basis of different types of addresses; on the physical layer using low–level physical addresses according to the hardware devices used, on the data link to presentation layer on a first–level abstraction using host addresses such as IP addresses[2], and on the application layer on a second–level abstraction using high–level, pronounceable hostnames.

The task of naming hosts and network domains is addressed by creating a hierarchical relation between domains, with hosts as the furthest descendants from an artificial root domain. By appending the domain labels one after the other to the host labels on the path up to the root in the hierarchical tree, a unique, memorizable, and usually pronounceable identifier is created: the hostname. One of the management tasks in the Internet is the mapping of lower–level addresses to these hostnames.

The mapping, or binding, of IP addresses to hostnames became a major problem in the rapidly growing Internet. Note that this paper does not deal with the mapping between addresses on the physical layer and transport layer, which is solved by ARP[3] in the TCP/IP Internet Protocol Suite, but with the mapping between hostnames and IP addresses.

[2] "32-bit addresses assigned to hosts that want to participate in a TCP/IP internet" [Com91]

[3] "Address Resolution Protocol – used to dynamically bind a high–level IP address to a low–level physical hardware address" [Com91]

This higher–level binding effort went through different stages of development up to the currently used Domain Name System (DNS). The DNS is a distributed naming resolution system used by most network services available throughout the Internet. It works transparently for the user who sends email, accesses another host via *telnet* or *rlogin*, or transfers some files via *ftp* between hosts. The DNS provides name binding in both directions: given a hostname, it returns the appropriate IP addresses, and vice versa.

Before hosts grant network services to users, an authentication process takes place, where the users' access rights, and the identity of connecting hosts get scrutinized, according to provider policies. There are many notions on how access rights can be specified. Examinations can be based on identification by hostname, login name, and login password. In some cases it suffices to provide the right names, and access is granted without specifying any password at all.

Some Berkeley *r–commands* (see [Ste90, chapter 14]) offer network services for which it is sufficient to verify user name and hostname to gain complete access. As the remote user name is specified by the connecting site, the authentication is additionally based upon the name of the connecting machine. A machine that offers services can acquire information about the *socket* that is used by the connecting site. A socket is an abstraction for a network service access point (NSAP): in UNIX[4] a tuple consisting of IP address, port, and protocol used by the remote site. To verify the hostname, it is the task of the DNS to map the IP address to the hostname.

Because the DNS is distributed among many thousands of hosts, it can be a critical mistake to blindly trust the resolved binding. This paper investigates policies and mechanisms to solve the problem of trust in the Domain Name System. Some of these policies and mechanisms might be abstractable to distributed naming services in general.

Although this problem has been known for some years now, not many publications deal with it. [Bel90] and [Sch93] are the principal accounts that we can mention as related work. [Bel90] demonstrates the subversion of system security using the DNS and discusses possible defenses against the attack and limitations on their applicability. The paper follows suggestions from Paul V. Mockapetris, the designer of the DNS. In [Sch93] the details of the exploitation of the weakness are worked out and several approaches to solve the weakness in the DNS are discussed with emphasis on hardening the name server implementations and the usage of strong cryptographic methods for authentication.

[4]UNIX is a trademark of Novell

2 The Problem

2.1 Statement of the Problem

Authenticity is based on the identity of some entity. This entity has to prove that it is genuine. In many network applications the identity of participating entities is simply determined by their names or addresses. High–level applications use mainly names for authentication purposes, because address lists are much harder to create, understand, and maintain than name lists.

Assuming an entity wants to spoof the identity of some other entity, it is in some cases enough to change the mapping between its low–level address and its high–level name. That means that an attacker can fake the name of someone by modifying the association of his address from his own name to the name he wants to impersonate. Once an attacker has done that, an authenticator can no longer distinguish between the true and the faked entity. This describes the fundamental problem on which this paper is based: *If the binding process between names and addresses cannot be trusted fully, no one can rely on an authentication process on a high–level.*

2.2 The Problem in the DNS

To understand the method how to deceive the DNS we first give an example for a valid name resolution in the DNS. The resolution is based on the client–server paradigm. Any process that accepts a connection from another host receives from its lower protocol layer the connecting host's IP address. The process then calls its local resolver with this IP address as an argument and requests the according hostname. The resolver forms a query for the given IP address and waits to retrieve the response containing the answer to its query from the default name server. This name server could be running on the same host with the resolver software, on a host in the local domain of the resolver, or on a host outside the local domain. The selection of which name server to contact depends on the name or address to be resolved. The decision process about this choice is specified in [Moc87, sections 4.3.2, 5.3.3].

Queries to name servers from a resolver come in two flavors: *recursive* and *iterative*. In recursive resolution, a resolver sends a recursive query to a name server. The queried name server then has the obligation to respond with the answer to that query or an error code. If a name server cannot resolve the query locally, it calls its resolver and queries recursively another name server. This is repeated until one queried name server supplies the answer or an error code that then travels the reverse path. In iterative resolution, the contacted name server returns an answer to the query to the requesting resolver. This is a referral to another name server that is more likely to know the answer, or an error code to signal the occurrence of an exception or error. The repeated resolution attempts are performed by the local resolver.

Many security problems of the TCP/IP protocol suite build on the ability of the attacker to spoof the IP address of a trusted machine, as described in [Bel89]. As

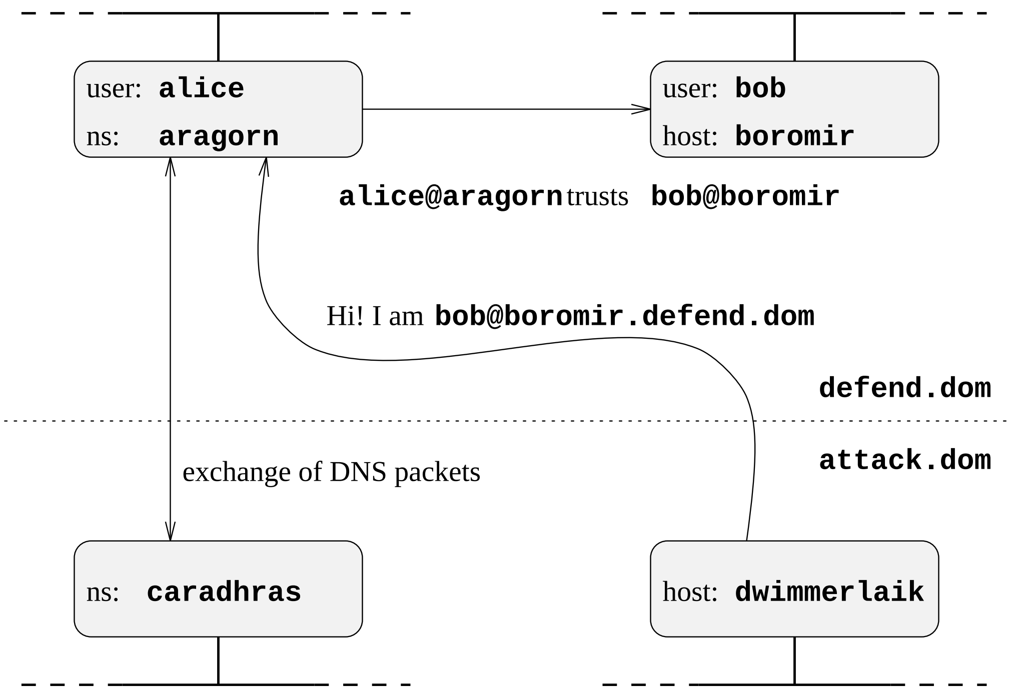

Figure 1: Example topology of machines

hosts trust each other, usually on the basis of hostnames, an attacker can take the easier approach and spoof a host's name instead of its IP address. The process is depicted in figure 1.

Assume that user `alice@aragorn.defend.dom` trusts user `bob@boromir.defend.dom` via the *.rhosts* mechanism. If a host named `boromir.defend.dom` accesses another host named `aragorn.defend.dom`, host `aragorn` accepts the connection and retrieves address information about the connecting host `boromir`. Host `aragorn` reads host `boromir`'s IP address and converts it into a regular hostname. To bind the right name to the IP address, host `aragorn` starts a DNS query in the reverse lookup tree, the database portion that contains the IP address to hostname mapping information.

For a pair of machines `caradhras.attack.dom` and `dwimmerlaik.attack.dom` under the power of an attacker, with `caradhras` running a primary name server for a certain zone, and `dwimmerlaik` trying to fake `boromir`'s identity, it is easy to make `aragorn` believe `dwimmerlaik` was `boromir`. `dwimmerlaik` connects to `aragorn` and claims to be `boromir`, `aragorn` retrieves `dwimmerlaik`'s IP address `111.22.33.4` and queries the name `4.33.22.111.in-addr.arpa` from the DNS. One single entry in the authoritative data for the reverse lookup tree for `caradhras`'s zone specifies the IP address–to–name mapping between `4.33.22.111.in-addr.arpa` and `dwimmerlaik`. If the attacker replaces this line by a mapping between `4.33.22.111.in-addr.arpa` and `boromir`, `aragorn`'s resolution attempt will finally grant `dwimmerlaik` access to `aragorn`.

This shows the simplicity of an attack that is based upon trust placed in the data provided by DNS. It is based on a weakness in the DNS, not an easily fixable bug in the implementation of a particular network service.

One widely accepted way of dealing with this problem is adding an additional DNS query of the determined hostname to the server code and comparing the returned IP addresses against the original IP address for a match. This only adds marginally to the quality of security; it does not provide complete security. An attacker can piggyback additional resource records to the answer packet to the first query. Doing so, the attacker poisons the victim's cache with false information, such that the forward lookup would not disclose the attack.

2.3 Weaknesses

In this paragraph we describe the conditions that facilitate a break–in. The DNS is weak in several places. We examine the problems of name–based authentication processes, trusting information that comes from an untrustworthy authority, and accepting additional, possibly incorrect information that was not requested, but that seems to provide advantages for runtime performance.

2.3.1 Assumptions to Facilitate Break–ins

In our setup we assume that the attacker has complete control over machine `caradhras.attack.dom` running a legitimate primary name server for a DNS zone. This strong assumption does not always need to be satisfied. It is simply the easiest way for an attacker if he controls a primary name server, because of its capabilities and the fact that other machines believe name servers.

Depending on the topology of a real network it is sufficient if an attacker controls one of the authoritative name servers for the particular zone: the one that is queried first by the remote resolver. It is not much more difficult for an attacker to satisfy this second assumption than the first one.

The control must include the ability to update the associated inverse mapping tree. The attacker might have successfully subverted such a machine or simply be the legitimate owner of it. In the following discussion we will assume that the attacker has such access to a primary name server.

2.3.2 Authentication via Hostnames

We explained in the introduction that users need to be authorized by network service providers before they can use the service. This authentication is usually based on the verification of the user's login name along with the associated password and the hostname of the machine on which the user starts his requests. Networks (as well as systems in general) may be classified into different partitions: Closed Networks, Open Networks, and Trusted Networks [PL91].

Closed Networks can be accessed only within certain boundaries. Sessions are controlled and secured in accordance with the rules implied by an organization's policy. In a Closed Network, the locations of all resources are well known and specified.

Open Networks are regions separated by boundaries from their surroundings, but the transfer of information across these boundaries is allowed. They are augmented by publicly accessible parts or connections to networks owned by other companies or organizations. These two extensions make this type of network vulnerable to external threats.

Trusted Networks introduce the concept that network access is controlled at the entry node. In the case of large international networks, maintainability and controllability are important issues. Adopting the Trusted Network concept allows the decomposition of a large network, growing towards an unmanageable complexity, into relatively small national or regional networks, each supported by local staff, and each provided with its own network access control. The advantages are increased controllability, maintainability, manageability, and simplification of change management. A Trusted Network can be regarded globally as a single Closed Network, but from a local point of view, the interconnected networks stand widely open with all the applicable security threats.

The Internet is a system of Trusted Networks within Open Networks. This allows the danger that once someone has falsely gained access to one machine, it is much simpler to subvert others. The term *net–surfing* describes the journey through a number of subverted systems with the goal of subverting others. Within Trusted Networks users are authenticated solely by their login name and connecting hostname. The login name is specified by the connecting site, and therefore can be falsified, such that the only reliable information left for the addressed machine is the connecting machine's IP address. The addressed machine then maps the IP address into a hostname using the DNS. If an attacker manages to subvert this name binding call, he can falsify the name of a machine within the Trusted Network and therefore succeed in his attack.

2.3.3 Trusting a Not Trustworthy Source

Using the DNS to map the IP address provided by lower–level protocol layers into the applicable hostname, the addressed host blindly trusts the information that is provided by the DNS. Information that comes from sources outside of the trusted area is trusted. That is a severe violation of the partitioning concept. Only truly authoritative information should be trusted.

2.3.4 Believing Additional, Not Authoritative Information

Efficiency is one of the stated goals of the DNS. The DNS protocol packets contain an additional answer section. Using this, name servers can provide resource records containing information that could be useful in future requests, but that were not explicitly requested. There are situations where these additional records aid system efficiency. If the answer to a query is a referral to another name server, then it is beneficial to add that name server's IP addresses to the response. That saves the lookup of the name server's associated IP addresses, once its name is found. Additional resource records are cached for future use.

As we rely on the correctness of these additional records once we use them, we trust information that comes from a source possibly outside of the trusted scope. That is another violation of the partitioning concept.

3 Policies and Mechanisms as Solutions

We identify policies and mechanisms that serve as solutions or that simply augment the level of security of the authentication process. Because many factors contribute to the security breach encountered in this paper and all of them are necessary for the weakness to exist, it is sufficient to eliminate at least one of them. That sounds easy to accomplish, but is a difficult task in practice, because eliminating any one of the factors brings with it a disadvantageous trade–off with functionality, efficiency, or convenience.

We describe evaluation criteria and present for each of our solutions necessary additional background, followed by a description of the idea of the solution. We make the distinction between mechanisms that enable the implementation of policies and solutions that consist solely of the implementation of a certain policy. Each solution is examined and discussed using applicable evaluation criteria.

It is important not to view these solutions as stand–alone. In different combinations they achieve several degrees of security. It is a good idea to implement a combination of the presented solutions, to obtain a greater level of confidence in the security of the DNS.

3.1 Evaluation Criteria

In solving the problem we are striving for *compatibility with the original design goals*. In the case of the DNS these goals are *data consistency* (to provide a consistent view of the name space to be used to refer to resources), *efficiency* (to handle the immense volume of data and resolution requests), a *distributed character* of the implementation (to provide fault tolerance and distributed authority and maintenance), *generality* (to provide a general usefulness that satisfies pragmatic reasons like implementation costs and administrative effort), and *independence* (to provide a portable system that does not depend on underlying hardware or communication technology.) Each of these goals represents a criterion in itself. Indeed, the ultimate goal is to guarantee data consistency, but not only in the data base but also during the mapping process. That means that we want to prevent the possibility of malicious software introducing wrong associations without the data base ever seeing changes. The correctness of this run time behavior is much harder to ensure than the integrity of the data base.

We consider the *quality of a solution* to be a measurement of the radius of applicability of the solution. The *feasibility of an implementation* of a solution determines how much effort is needed to apply the solution to an unmodified version of a state–of–the–art name server. The *complexity of its implementation* denotes if modifications in different areas are involved and how complicated their interaction is. Solutions might not be suitable in every organizational environment. We call this criterion *applicability in an organization*. The *transparency of the solution* involves the software interface and the user interface to the system. A solution that does not require changes to the DNS protocol is preferable over one that does. User approval of any modification that is not transparent is a crucial point. We combine these aspects in the term *acceptability by the user*. An important point in the introduction of changes to systems is the *transition process* from the original state (before the solution is applied) to the new state.

3.2 The Berkeley Patch

We briefly explained the Berkeley software patch in section 1 without calling it the Berkeley patch. This first attempted defense, developed at the University of Berkeley, CA , consists of modifications of the r-command daemons. The idea is to validate the inverse mapping tree by looking at the corresponding node on the forward mapping tree. S. Bellovin describes the method used by the patch in [Bel92] as follows:

> To detect this, we perform a cross–check; using the returned name, we do a forward check to learn the legal address for that host. If that name is not listed, or if the addresses do not match, alarms, gongs, and tocsins are sounded.

The fix is easily installed and not very complex. Its compatibility with the existing DNS protocol is another advantage. The transition process to move to services that contain the patch is not difficult, but requires some work. Although we regard this patch as an obligatory modification to daemons like *rlogind* and *rshd*, it is limited in its scope. The cache of a running name server can still be poisoned by supplying additional unrequested records as the experiments described in [Sch93, section 3.5] prove.

The Berkeley patch utilizes a principle that can be applied outside of the UNIX domain. The idea is to perform a cross-check of the first mapping in the reverse order. In a consistent state, forward and backward mapping data are managed by the same authority. Thus tampering with only one of the two directions of mapping can be detected.

The patch is a solution if trust can be extended only within the scope of authoritative data, and if the attacker does not use the more sophisticated attack method. If the attacker supplies the additional address record with the answer to the reverse lookup, it means that he controls both lookup directions, and that trust is extended to possibly untrustworthy sources.

3.3 Examining Berkeley *r–Commands*

In this paragraph we discuss the UNIX–specific way of implementing a Trusted Network. The Berkeley r–commands extensively use the *.rhosts* and */etc/hosts.equiv* files to increase convenient network access. In paragraph 2.3.2, we discussed the Trusted Network concept. R–commands such as remote login and remote shell offer the possibility to extend trust to other machines. Users and system administrators can build individual networks of trust. This proves dangerous in some cases. [GS91, chapter 11] discusses security problems with the UNIX trust mechanism.

The existence of these structures of trust is necessary for the break–in to happen. Obviously, the break–in is prevented if we prohibit the usage of trusted hosts or trusted users completely. It is technically possible to disallow the usage of *trust* in Berkeley r–commands. The choice can be made by the system administrator at compile time. However, being able to access other machines without passwords makes the work in a networking environment easier. Once used to the comfort, not many users agree to sacrifice their convenience for the prevention of *hypothetical* security concerns. The trade–off hereby would contain the loss of convenient, and in many cases, necessary tools for trouble free connection to hosts that are accessed frequently.

A less safe solution would be to limit trust to locally administered zones, i.e. authoritative zones, where the Berkeley patch works reliably. As we discovered in paragraph 3.2, limiting trust to certain zones fixes the flaw. An organization could issue the policy that only local trust is allowed. In some organizations this can be considered a reasonable approach if hardly any remote accesses that are directed to hosts in the local zone are originated outside of the local zone. Additional mechanisms would be necessary to enforce the policy, such as periodical checks of *.rhosts* or a modified r–command implementation where users cannot directly modify their database of trusted machines, but have to use a special program. The trust associations must then be kept in a protected data area of the operating system. This program could filter out–of–zone entries at the time the user wanted to enter them. It would also contain the possibility of managing setup changes centrally. This solution actually proposes an automatized procedure to implement an organization's policy.

If the nature of connections allows a policy such as described above, implementing it is a major effort. Some system scripts have to be written to ensure proper usage, operating system code and r–command code must be modified, and a new user interface has to be developed. Users have to be trained on how to apply the changed facility and have to be made familiar with the new policy and the new user interface. Advantages of this new approach are compatibility with the existing DNS protocol and additional benefits in further security related issues.

Although we concentrate on the Berkeley r–commands in this paragraph, we do not forget that there are other ways to exploit the flaw. For example, intercepting electronic mail is a target of attackers; especially electronic mail that is exchanged by security agencies and security related organizations. Electronic mail depends on the DNS.

The Massachusetts Institute of Technology, together with IBM and Digital Equipment Corporation developed in 1983 Kerberos, an authentication system that uses Data Encryption Standard (see [NBS77]) cryptography to transmit sensitive information on a network, such as clear-text passwords. Although Kerberos is an excellent solution to several difficult problems, it has shortcomings that limit its usefulness in respect to our problem. A discussion of its shortcomings can be found in [GS91].

Overall, a very weak point in Berkeley derived UNIX systems is the usage of trust. This paper exploits only one of several known flaws based upon trust. Using trust–based mechanisms requires thinking about a change in individual policies in dealing with granting trust to others. We can conclude, by citing S. Bellovin ([Bel90]):

> If a host trusts another host not named in a local zone, its name server cannot protect it.

3.4 Restricting Public Information Access

What makes the break–in possible in the first place is gathering necessary information about hostnames of trusting machines and user names on different systems trusting each other.

We are not discussing random patterns of trust that might exist between hosts, but common patterns using a systematic approach. In a cluster of time–sharing machines, each machine is likely to extend trust to all its peers. This pattern is not common to the general user population, but it is applicable to systems programming and operational staff. Another typical pattern is the occurrence of file servers that trust their clients, who serve as a source of extra CPU cycles. Dataless clients will frequently trust administrative machines to permit software maintenance. Some systems still use the same */etc/hosts.equiv* files on many hosts just to simplify systems administration.

Generally accessable programs can aid in discovering the desired information: there are network monitoring and information tools (such as *snmptnetstat*, *traceroute*, or the DNS itself), user information services (such as *finger*), and UNIX services in general (such as *ftp*, *smtp*, or *rpcinfo*.) Other sources of information might include published material describing network topology that is available for example from some academic departments.

The mentioned collection of tools shows that it is a difficult task to limit information access without sacrificing the legitimate utilization of network services. Preventing someone from gathering information is nearly impossible. Too many services rely on address information, and we conjecture that most users would not be happy if they were deprived of useful tools such as electronic mail or news readers. The idea of open systems requires open access to information services and address information. Therefore, most system administrators have decided that the benefits of these utilities outweigh the risks.

3.5 Adjusting DNS Update Intervals

Some sites have connections chiefly with machines outside of their zones that stay stable in the sense that hostname to IP address mapping will stay the same for a long time. The idea is to enter long time–to–live values into the resource records, values that exceed the currently implemented threshold of 1 week. Limits could be increased up to 6, 12 months, or even longer, depending on the situation. If this data is entered with great care to ensure correctness of the mappings, the DNS based break–in is prevented.

This approach is limited by its scope of applicability, but it is a solution with many advantages. It goes with the current DNS protocol and can be implemented without much effort by simply changing the constant in the name server code that determines the maximum time–to–live for cache entries and recompiling the system. As all necessary entries are kept in the local cache, the system provides very quick replies to queries. It hardly ever uses the network and therefore saves bandwidth on the medium for other tasks.

This approach has the problem of validating mappings before they are cached. How can it be ensured that the mappings are correct in the first place? Certainly, a false entry would stay for a long time, and the attacker's address would be finally noted. But does that really help, once mischief is done? It might aid in prosecution efforts, but only little in prevention.

Extending TTL values to a long period of time is a safe and feasible method in environments where the additional condition of static mappings with long lifetimes is given. However, in this scenario the DNS seems not to be the right approach, but a locally well–administered static mapping mechanism.

One of the original reasons to introduce the DNS was to manage the dynamic behavior of changes in the data base. This approach fixes mappings for a long time and uses a powerful distributed database system for an infrequently occuring update process. Although we are not talking about a static mapping in this paragraph, a well–maintained *HOSTS.TXT* file or a hybrid approach would have the functionality required with less overhead.

It could be suggested to abandon the DNS and either return to the previous system with a static host table, or move on to another system that has yet to be developed. We are not going to discuss possible future development of the DNS here, but returning to the previous system.

In this approach, mappings can change frequently, but changes have to be reported to a central authority that manages the whole DNS space in contrast to the DNS approach of managing zones through delegated local authorities. This would not solve the problem, because the problem is not the DNS, but inadequate methods of host authentication. IP addresses of trusted machines could still be imitated. This is a somewhat harder task, but the techniques have been known for quite some time (see [Mor85]).

Would it be safer to transmit updates to a central site? Electronic mail, telephone calls, or conventional paper are not necessarily a reliable way to transmit mapping information updates. The long time delay until centrally made changes are propagated through the network would condemn the database to be in an inherently inconsistent state. The system would again contain all the disadvantages which were the reasons for developing the current DNS.

But besides these obvious, technical, and well–known reasons, there is a significant argument why no one can possibly be in favor of reinstalling the previous system: the sheer size of the Internet. *HOSTS.TXT* was abandoned because 200,000 hosts was too much to be managed. Are currently over 2.2 million (see [Lot94]) easier to handle? Certainly not.

Abandoning the DNS would drag the name resolution task in the Internet out of a functioning state with a not easily exploitable security breach, into an unmanageable, not working state of prehistoric system design. We think that would do more harm than ignoring the problem.

3.6 Hardening Name Servers

3.6.1 Keeping Additional Information

A first idea is to extensively log remote login attempts with all associated address and name information. Or even more: to tag cache entries with their origin. The latter is an easily achieved modification that costs additional memory space in the cache. This method makes it easier to track false database entries for the purpose of debugging wrong zone data or investigating a DNS based break–in.

3.6.2 Prevention of Cache Poisoning

Preventing the cache from contamination is not feasible from within the name server code, as there is no way of a priori determining if any given additional record is trustworthy or not. We could start treating special cases of when to allow or disallow additional information.

The default safe behavior would be to disallow the caching of unrequested information, and to allow it only in cases where the information is necessary, and then only for the current resolution.

3.6.3 Context Cache

There are other, more sophisticated approaches possible: if some additional or authoritative records are returned together with a resource record, they could be interpreted only in the context of that resource record. The difference between the default safe behavior approach and this one is that in the former, resource records are only cached when they were requested or necessary additional information, whereas in the latter approach the new entries get cached, but can be retrieved from the cache only in the same context in which they were entered. For example, an `address` record in the additional section of a response to a `mail exchange` record request should only be used for delivering mail. The information would not be acceptable for a remote login to another host, or generally usable for other services. A glue `address` record coming along with a `name server` record would only be used for follow–up queries, because that is the context in which it was supplied. `Address` records along with `pointer` records should never be cached, because there is no legal context in which they have to be returned in a single response.

This whole approach leads to the question of whether we still need the additional section at all. If only certain combinations of resource records are allowed as a response to a query, why not consequently eliminate the idea of additional, unrequested information completely, and adapt the protocol to accommodate the new ideas, namely a certain limited number of types of associations?

First of all, that would require a protocol change, which is something we try to avoid. Some of the original design goals of the DNS also imply that eliminating the additional section would not be a good approach. The system would lose some of its generality, because the additional section might become very useful in future applications of the DNS without containing any security threats. The system would certainly lose efficiency. Here we see again an important trade-off that we have already mentioned in previous sections: an increase in systems security and a decline in system performance vs. good system performance and a possible lack of security.

It is therefore justifiable to take the approach of hardening the name server by treating more special cases, and by increasing the complexity of the internal data bases, instead of hardening it by implementing the same ideas accepting protocol changes.

3.6.4 Authority Cache

A further approach would be to cache data only if the source of a record is known to be authoritative for that zone. We give an example for that: If a name server `aragorn.defend.dom` receives a `pointer` record from some host `caradhras.attack.dom`, and the DNS message also contains an `address` record in its additional section, then the name server `aragorn` would believe and cache this information only if it already knows that the source name server `caradhras` is authoritative for the according zone. A name server following this strategy would create its own tree of authoritative name servers. This tree would have to lose subtrees according to the expiration of the lifetime of some node (name server).

This approach however has a serious flaw in it. Servers determine if DNS messages are genuine by checking a certain flag in the header of the DNS message: the `authoritative answer` bit. This flag is only valid in responses and specifies that the responding name server is an authority for the domain name in question. Nothing prevents any attacker who supplies specifically manufactured packets in the first place from setting this bit regardless of its validity.

3.6.5 Conditional Cache Use

The Berkeley patch (see paragraph 3.2) can fail in the case that the cache is already poisoned. An idea to strengthen the Berkeley patch is to provide the possibility to resolve queries without using the cache. That could be used by the Berkeley patch. The system call executing the forward lookup would for example set a flag to indicate that the cache contents should not be used for the following resolution. This method again decreases the efficiency of the system, but it prevents the exploitation of the weakness. One could also think of a system call to flush the cache followed by a reload of the database, similar to the signal SIGHUP that a system administrator can send to the BIND implementation of the name server to achieve the same.

3.6.6 Discussion

A very thorough analysis of the protocol is needed to determine the cases in which additional resource records are legal and cannot do any harm, or have to be stored in different contexts.

One of the design goals of the DNS is hereby in danger: generality. The DNS should not contain any unnecessary restrictions regarding its purpose or applications. If the implementor of the DNS were to decide which combinations of resource records would be allowed, the DNS might be constrained in a way that it is no longer useful for certain applications. A decline in system performance would result from the fact that name servers would believe and therefore cache less data — data that might be needed later.

Hardening name servers consists of several possible modifications, some of which seem promising, even though their application decreases the system's performance and increases its complexity.

3.7 Cryptographic Methods for Authentication

In this paragraph we describe the architecture of an authentication system embedded into the DNS. Note that the algorithms and methods described in the following paragraphs yield as much security as possible. However they are not perfect. Most of the algorithms rely at some point on conjectures in number theory that are neither proven nor contradicted, or on the fact that brute force attacks are computationally infeasible. For a discussion of this see [Den82].

We have to meet the requirements of data integrity of the message and of originator authentication. In the following we will elaborate on these two requirements and present techniques for their possible implementation. The algorithms and cryptosystems that we chose are typical representatives of the class of algorithms that are applicable. They are by far not the only possible choice.

3.7.1 Data Integrity

Data integrity in a communication system prevents against active wiretapping, that means a recipient is provided with the assurance that the content of a received message is identical to the content of the message sent by its originator.

We want to ensure the integrity of transmitted DNS messages along with a time stamp to protect against replay attacks. We concentrate on a certain technique to detect unauthorized message alteration that is efficient and considerably secure.

In case of alteration detection, recovery actions could be to ignore the received DNS message and issue an additional query. Our approach is based on message digest algorithms. Message digests, or synonymously fingerprints or signatures, are the result of the application of a one-way hash functions that computes a checksum of its input data.

MD5 and the Snefru algorithm are examples for message digest algorithms (see [Riv92, Mer89].) Message digest algorithms are easy to compute, are only a few bytes per message, are computationally hard to invert, and usually require a certain size of input data.

An originator would calculate the message digest of a DNS message immediately before it is sent out. The recipient would recalculate the message digest and compare the resulting value with the one calculated by the originator. In case of a mismatch, the receiver would conclude that he received a modified DNS message. He would discard it.

But how does the message digest calculated by the originator reach the receiver without modification? The message digest algorithms are publicly known and anyone tampering with a message could easily modify the associated message digest accordingly. To show how this can be prevented we discuss a method for originator authentication in the following paragraph. Message digests together with originator authentication give a very strong guarantee for the detectability of active wiretapping.

3.7.2 Originator Authentication

Originator authentication permits the recipient of a message to reliably determine if the originator of a message is who he claims to be.

We explain briefly a procedure that guarantees the originator's authenticity. In an asymmetric cryptosystem a pair of distinct but mathematically related keys is used for

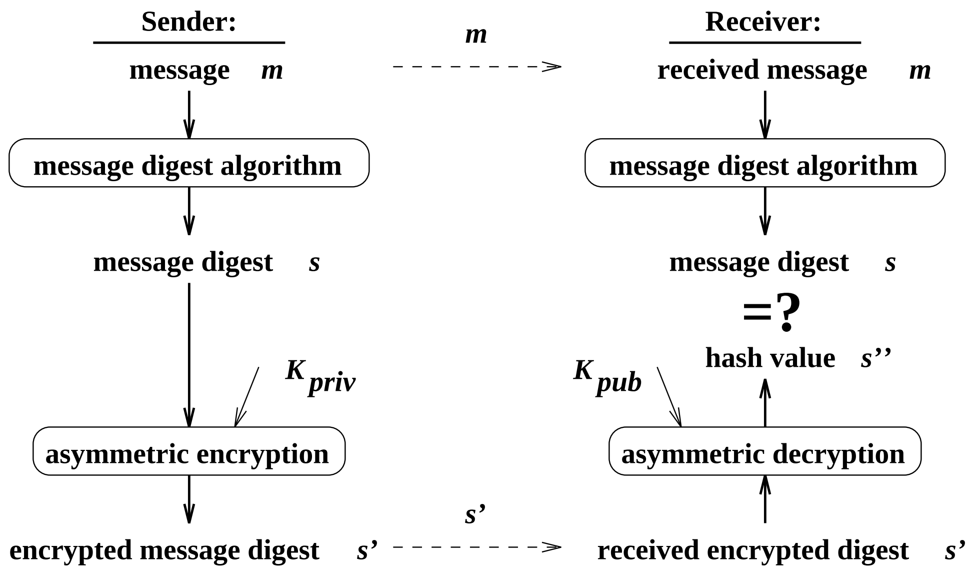

Figure 2: Digital signature generation and validation

encryption and decryption. One key is private and kept secret by the sender, the other one is publicly known. Data encrypted with a sender's private key can be decrypted using his public key, and vice versa. These keys are usually large integer numbers, several hundred decimal digits long with special, mathematical properties. *Pohlig-Hellman* and *RSA* are examples of asymmetric cryptosystems (see [PH78, RSA78]).

Figure 2 depicts digital signature generation and validation; a more detailed explanation can be found in [Sch94, section 17.6]. The sender digitally signs data m by encrypting the hash value s of the data using his private key component K_{priv} and sends $(\mathrm{E}_{K_{priv}}(s),m)$. The receiver validates the data in a three step process. He computes the hash value s of the data m, decrypts the hash s' that arrived using the signer's public key K_{pub} and compares the results $\mathrm{D}_{K_{pub}}(s')$ and s.

Why do we calculate a message digest at all and not simply encrypt and then transmit the whole message? The main point here is the difference between the runtime costs of creating a message digest and encrypting a message, depending on the length of the original message.

Runtime costs for public key encryption are rather high. Many CPU cycles are needed. Therefore we want to reduce the size of the data portion that has to be encrypted: in our case the output of the message digest algorithm.

Runtime costs for the hash functions are rather small compared to those of public key encryption. It is therefore important to note that it is more efficient to pad a short DNS message, calculate its fingerprint, and then encrypt the fingerprint, than simply to encrypt the whole DNS message. Message digest lengths are generally shorter than typical DNS messages.

3.7.3 Passing Credentials to Prove Authority

The crucial point in the previously described protocol is the importance of the public key of the sender. If an attacker can convince the receiver to use key K'_{public} instead of K_{public}, whereby the attacker possesses the related $K'_{private}$, the attacker can subvert the protocol such that the receiver will be fooled into accepting the integrity and origin of the message. This demonstrates that it is important to devise a scheme that protects against this threat. We solve this problem by the implementation of a distributed scheme for the validation of public key component certificates.

The name server sending the DNS message has to provide credentials signed by its parent domain, to convince the recipient of its authority over the domain for which it just resolved a mapping.

The use of such a certificate transforms the problem of establishing the credibility of one entity into the problem of establishing the credibility of the entity issuing the certificate. This problem is very closely related to the problem of distributing public key certificates. The CCITT recommendation X.509 shows a way to solve this problem. In X.509, a certificate binds a public key to a directory name and identifies a party that vouches for the binding.

We can adopt this mechanism, such that a certificate binds all name servers that are authoritative for a certain zone to this zone of authority and identifies the zone that vouches for the binding. X.509 imposes no constraints on the semantic or syntactic relationship between a certificate issuer and a subject. However, in our approach, the certification system takes the form of a single rooted tree. Each node represents a zone. Several name servers serve as certification authorities for each zone, because all servers that were introduced to increase the reliability of the database system are capable of valid and authoritative referrals.

A certificate for a zone consists of all IP addresses of authoritative name servers for that zone, signed with the private key of the name servers for the parent domain. Any resolver that receives a DNS message receives as part of it this certificate. After obtaining the public key for the parent zone of the queried zone, the resolver can then verify the validity of the referral. But to verify the authority of the parent zone, the resolver has to ask this zone for credentials.

This validation process for certificates is done recursively up the zone hierarchy tree that coincides with the certification hierarchy, starting at the name server that provides the queried mapping. The recursion will stop at some point, either at the root, or at some intermediate node that was certified before. The certificates that a name server holds are subject to timeouts, just like the resource records that specify bindings of this name server. The certificate for the root must be transmitted by some trusted, out-of-band mechanism. For example, the root certificate could be published in an international newspaper.

Even if an attacker manages to get a valid certificate of a name server it wants to impersonate, and has the capability to also spoof this name server's IP address, it is still not possible for the attacker to impersonate another host. As we saw in the previous paragraph 3.7.2, a DNS message is encrypted with the name server's private key before it is sent out. The credentials are part of the message and are therefore also encrypted. An attacker cannot construct the correctly encrypted message without breaking the asymmetric cryptosystem used.

3.7.4 Discussion

The validation of integrity and originator of the message, and its underlying pattern of certifications stating trust, are the features that make this approach secure. The following discussion shows its disadvantages. Some of them are serious enough to restrain from an implementation of this approach at the current time.

The whole procedure is time and space consuming. Many rather long public keys have to be stored (at least 200 decimal digits long each to make the public key encryption reasonably strong.) Obtaining memory for them, as well as additional cache memory for larger resource records, is not a problem in current architectures. The keys must be obtained before they can be used. S. Kent describes in [Ken93] certificate based key management for usage in Privacy Enhanced Mail (PEM).

We will not go into more detail regarding the key distribution process. The registration process that has to occur out–of–band is rather cumbersome. The calculations to encrypt and decrypt message digests may take too long to support the efficiency goal of the DNS. The additional data that has to be transmitted would not degrade performance too badly, especially if faster transmission media becomes broadly available, but the calculation overhead for encryption and decryption cannot easily be amortized. However, the *RSA* cryptosystem is available in hardware and a dramatic performance increase can be observed, compared with a software implementation of the same algorithms.

The implementation of such a solution is a major effort. The whole key management problem is complex and it also requires additional administrative effort. Resolver routines and name server routines have to be modified, along with the DNS protocol. The implementation is feasible, though very complex. Another drawback is the transition phase that is necessary because of protocol changes. Decreased performance because of calculations necessary to sign, encrypt and decrypt messages would be noticeable by users and real-time applications.

Currently, the method seems to be infeasible, because of its large computational overhead. Further drawbacks are the necessary protocol changes and the complexity of proper key and certificate management. However with further advances in processor speed and some reasonable relaxation on requirements for strong encryption (i.e. shorter keys increase performance of *RSA* dramatically) this approach can become very attractive in the near future.

4 Conclusions and Outlook

Where host identification is part of the authentication between communicating entities the validity of the authentication process can only be trusted as much as the resolution process that supplies the bindings between high–level hostnames and low–level host addresses.

This is a significant problem, because it exposes probably hundreds of thousands of hosts that are currently connected to the Internet to the threat of break-ins.

We discussed solutions to the problem with the concrete instance of the Domain Name System. We stressed hardening current implementations of the name servers and put emphasis on the development of a future scheme that uses cryptographic methods to give a strong guarantee for detection of spoofed bindings.

Acknowledgements

We would like to thank COAST sponsors BNR, Trident Data Systems, and the US Air Force, and the Fulbright Commission for support that aided, in part, this work. Thanks to Steven Bellovin whose valuable comments are most appreciated, Dan Trinkle who showed us how to master some of the subtle difficulties of the DNS, and J.R.R. Tolkien whose fantasy provided the hostnames.

References

[Bel89] Steven M. Bellovin. *Security Problems in the TCP/IP Protocol Suite.* AT&T Bell Laboratories, Murray Hill, New Jersey, April 1989.

[Bel90] Steven M. Bellovin. *Using the Domain Name System for System Break-ins.* AT&T Bell Laboratories, Murray Hill, New Jersey, 1990. (unpublished technical report).

[Bel92] Steven M. Bellovin. There Be Dragons. In *UNIX Security Symposium III Proceedings*, pages 1–16, Baltimore, MD, 1992.

[Com91] Douglas E. Comer. *Internetworking with TCP/IP.* Prentice-Hall, Englewood Cliffs, New Jersey, second edition, 1991.

[Den82] Dorothy E. Denning. *Cryptography and Data Security.* Addison-Wesley Publishing Company, Inc., 1982.

[GS91] Simson Garfinkel and Gene Spafford. *Practical UNIX Security.* O'Reilley & Associates, Inc. Sebastopol, CA., 1991.

[Ken93] Stephen T. Kent. *RFC-1422 Privacy Enhancement for Internet Electronic Mail: Part II: Certificate-Based Key Management.* Network Working Group, February 1993.

[Lot94] Mark Lottor. Internet Domain Survey Jan 94. SRI International, January 1994.

[Mer89] Ralph C. Merkle. Snefru. Xerox Corporation, Palo Alto, CA, 1989.

[Moc87] Paul Mockapetris. *RFC-1034 Domain Names - Concepts and Facilities.* Network Working Group, November 1987.

[Mor85] R. T. Morris. A Weakness in the 4.2BSD UNIX TCP/IP Software. Computing Science Technical Report No. 117, AT&T Bell Laboratories, Murray Hill, New Jersey, February 1985.

[NBS77] NBS. Data Encryption Standard. National Bureau of Standards, Washington D.C., Jan. 1977. FIPS PUB 46.

[PH78] S. Pohlig and M. Hellman. An Improved Algorithm for Computing Logarithms over **GF**(p) and its Cryptographic Significance. *IEEE Transactions on Information Theory*, IT-24(1):106–10, January 1978.

[PL91] R. Paans and H. de Lange. Auditing the SNA/SNI Environment. *Computer & Security*, 10(3):251–61, May 1991.

[Riv92] Ronald L. Rivest. *RFC-1321 The MD5 Message-Digest Algorithm.* Network Working Group, April 1992.

[RSA78] R. Rivest, A. Shamir, and L. Adleman. A Method for Obtaining Digital Signatures and Public Key Cryptosystems. *Communications of the ACM*, 21(2):120–6, February 1978.

[Sch93] Christoph L. Schuba. Addressing Weaknesses in the Domain Name System Protocol. Master's thesis, Purdue University, West Lafayette, IN, August 1993.

[Sch94] Bruce Schneier. *Applied Cryptography.* John Wiley & Sons, Inc., 1994.

[Ste90] Richard W. Stevens. *UNIX Network Programming.* Prentice-Hall, Englewood Cliffs, New Jersey, 1990.

[Tol65] John R. R. Tolkien. *The Lord of the Rings.* Houghton Mifflin, Boston, second edition, 1965.