A Distributed Systems Approach to Secure Internet Mail[*]

Philip Machanick
School of ITEE
University of Queensland, St Lucia
Qld 4072, Australia
philip@itee.uq.edu.au

Abstract

One of the obstacles to improved security of the Internet is ad hoc development of technologies with different design goals and different security goals. This paper proposes reconceptualizing the Internet as a secure distributed system, focusing specifically on the application layer. The notion is to replace specific functionality by an equivalent, based on principles discovered in research on distributed systems in the decades since the initial development of the Internet. Because of the problems in retrofitting new technology across millions of clients and servers, any options with prospects of success must support backward compatibility. This paper outlines a possible new architecture for internet-based mail which would replace existing protocols by a more secure framework. To maintain backward compatability, initial implementation could offer a web browser-based front end, but the longer-term approach would be to implement the system using appropriate models of replication.

keywords: Internet security, privacy, email, distributed systems, distributed file system

1 INTRODUCTION

This paper proposes an overhaul of existing Internet technologies by introducing a new infrastructure for distributed applications, starting from a secure distributed file store. Internet services were devised in a networked world, with the emphasis on communication protocols. Distributed systems start from the principle that the network is transparent to the extent that performance makes transparency possible [Tanenbaum and van Steen 2002], with emphasis on processes and data management. Network protocols form a lower-level layer, rather than an aspect of the user experience. The focus in this paper is on how the new model could be structured, as a starting point for defining a new Internet architecture. The proposed new model could then become the basis for a new approach to security. To make the discussion concrete, a possible replacement for email is outlined.

A big problem with existing Internet infrastructure is that it grew out of an era when there were relatively few users, who were a largely trusted community. For

[*]To appear in *Computers & Security*; copyright ©Elsevier 2005

example, `sendmail` was originally implemented to cover almost all aspects of email from user interactions to application-layer protocols. The `sendmail` program ran as `root` (super user), with various trapdoors for debugging [Landwehr et al. 1994] – an approach which could not be used in devising a new Internet application today. Further, mail has grown with ad hoc additions such as attachments, HTML formatting and scripting, all of which introduce security and privacy problems.

This paper proposes that the Internet be reconceptualised as a distributed system of loosely-co-operating processes, with a uniform model for document storage – whether they are seen as email, web pages or other more general content. The Internet can, of course, be seen as such a model in its current form, but the goal here is to redesign services so that they are a closer fit to a design intended to achieve security, while supporting further growth. The intent is that most existing services should be possible to reimplement in a more secure framework, with backward compatibility.

The approach here is to identify reasonable abstractions which could become the basis for a design, with some detail to show how it could be implemented. Specifically, a new model for mail is proposed, based on a distributed computing strategy. This new model could be implemented initially using a web front end, and gradually migrated to a more secure platform, which could become the base for more general services, in line with the proposed unified document model.

The remainder of this paper outlines the background to the problem, followed by a conceptual design. In conclusion, the practicality of the idea is evaluated, and possibilities for further work outlined.

2 BACKGROUND

2.1 Introduction

To provide a basis for the proposed design, it is useful to review some of the security and privacy problems with existing Internet services. These problems are largely a consequence of the fact that the Internet has grown organically from a starting point where security and privacy were not significant concerns, to a situation where it is a major infrastructure for commerce and personal communication. Further, it is useful to understand general approaches to distributed system design, as a basis for choosing suitable abstractions and principles for the proposed design.

This section briefly surveys known problems with relevant aspects of Internet security, then switches focus to distributed system models, specifically aimed at possible alternatives for popular Internet services such as email and the world-wide-web.

2.2 Internet Security

Internet security problems broadly speaking fall into two categories: insecure implementation of servers or clients (application layer), and insecure network protocols (lower layers).

Security presents a large number of problems at all layers of protocols [Bellovin et al. 2003]. The fact that a glossary of Internet security terminology runs to 191 pages [Shirey 2000] is some indication of the magnitude of the problem.

This paper focuses on replacing the existing insecure application layer, and assumes that security at lower layers will not be a more serious problem in the proposed model than in existing approaches. The focus here is therefore on security problems with clients and servers.

Some examples of application-layer problems include:

- buffer overflow problems – usually a result of overwriting the return stack to point to code to set up an attack [Park and Lee 2004]
- stealing passwords – e.g., by a dictionary attack [Pinkas and Sander 2002]
- worms – transmitted via networks; there have been times when major network outages have been caused by worms, and they have turned out to be one of the most ubiquitous security threats, spreading rapidly to new applications and platforms [Weaver et al. 2003]

If we focus our attention specifically on email, a widespread security problem arises from attachments, which can have malicious semantics. A big privacy problem arises with junk mail – spam [Sipior et al. 2004] – because once an email address is publicly known, it costs almost nothing to send to it. Further, since there is no authentication of senders, a spammer can safely send millions of copies of an email without fear of an (possibly) inadvertent denial of service attack on their server by angry recipients.

Attachments have other inconveniences. For example, if there is collaboration on editing a document, communicating updates in attachments results in multiple copies, which can be hard to manage.

While these problems can be addressed by ever-more sophisticated strategies for detecting worms [Zouand et al. 2004] and applying security patches, it is worth pursuing alternative architectures which avoid the underlying problems.

2.3 Distributed File System Models

There are many distributed computing models to choose from. However, it is useful to limit the options to those which are reasonably close to the existing Internet, to make backward compatibility a reasonable prospect. Specifically, models for distributed file systems are a useful starting point. While distributed processing models could play a role, for purpose of this paper, distributed file systems will provide enough principles.

A distributed file system has significantly greater latency problems than a purely local file system. For this reason, disconnected operation and replication are useful. Disconnected operation allows work to continue when the network goes down, with a reintegration phase when connectivity is restored [Kistler and Satyanarayanan 1992].

Managing access rights across a distributed file system requires mechanisms to grant and withdraw rights. One approach is the use of *capabilities*. If access rights are seen as forming an access control matrix (rows are rights, columns are objects), a capability can be thought of as a row of the table, representing a specific set of rights to a list of objects. The Amoeba distributed system implemented capabilities as a 128-bit number, including not only the identity of the object and its rights, but where to find it, and check bits to prevent forgery [Tanenbaum et al. 1990].

If capabilities are combined with authentication [Sirbu and Chuang 1997], it becomes possible to implement a distributed file store in which access control is managed with a small number of mechanisms, simple enough that verification becomes an option.

That is not to say that security in distributed file systems is trivial. There are many issues to be worked through, such as trust [Aberer and Despotovic 2001], which apply in a worldwide system available to all users, which add further to the problems already known in distributed systems design.

However, the fact that the problems can be isolated into one subsystem is an attractive advance on the existing model, where all clients and services potentially contain security holes (both in the client or server software, and in any other aspect of the system).

2.4 Summary

This section has briefly reviewed Internet security problems, and distributed file systems, to provide some background for a proposed approach to Internet applications.

The coverage is not intended to be comprehensive, but rather to provide a base for moving away from the current strategy to one where problems should be more contained.

3 THE PROBLEM

3.1 Introduction

The key issue being addressed in this paper is how to get the effect of existing Internet services in a consistent model with less potential for security holes. The approach is to reconceptualize existing services as a distributed system, rather than a loosely-coupled collection of protocols and services.

The less specific the model, the more likely it is to be general. Nonetheless, it is useful to make the design principles concrete with an example. The chosen example, email, is a ubiquitous service which increasingly interacts with other services (through HTML formatting, scripting and links to web pages), so it is a useful starting point. Principles and abstractions which work well with this example – provided they are not tied very specifically to email functionality – are likely to work in a range of other cases.

The remainder of this section outlines some features of email, and ends with a summary of problems with the existing model.

3.2 Mail Example

The intention here is to illustrate principles, rather than to arrive at a comprehensive design, so a few key aspects of email functionality are considered, including some features currently problematic. A full design would have to examine a range of typical uses of email, as well as a range of new possibilities opened up by a redesign.

The remainder of this section briefly outlines some aspects of email, then examines how the equivalent functionality could be implemented using a distributed systems model.

3.2.1 Mail Functionality

Email can be thought of at two levels: underlying protocols, and user-level features. Since the goal is to look at a reimplementation on a different infrastructure, the focus here is on user-level features.

Features common to a range or email clients include

- compose a message – increasingly often with features to support formatted text (usually HTML, though there are other formats in use, such as RTF)
- attach a document – often without constraints as to the document type. The effect usually is to create a new copy of a document embedded in the message, which has to be extracted when the message is received or read
- send a message – with or without facilities to queue messages to be sent later
- check for new mail – with or without periodic automatic checking
- fetch or mirror mail – depending on the client and server type, mail may be a local copy, or be removed permanently from a server
- view headers – with or without threading to group related messages

This feature list, while not exhaustive, is enough to isolate a few problems worth tackling.

First, composing a message assumes knowledge of the recipient's capabilities, unless plain text format is used. If nothing is know about the reader, a plain-text version may be necessary with the formatted version appended as an "attachment".

Attachments pose enough problems to itemize:

- copy semantics – should editing the original document after attaching it to an email, but before it is sent, result in the original or the changed version being sent? If you change a document after sending it, do you always remember to resend it to everyone who was sent the original?
- multi-author documents – sending email attachments to multiple authors working on a common document results in difficulties in tracking versions
- multiple copies – sending an attachment to multiple recipients results in possibly unnecessary duplication

It would be easy to identify more problems, but these should be enough to go on.

Sending a message conceals a potential problem many users run into: there is no way to unsend a message once it has left your system (or, if you're lucky, a server on your own network).

Checking for new mail does not expose any serious problems. However, it does illustrate one aspect of mail which is useful to preserve: its asynchronous nature.

Viewing headers in a summarized form again is a useful feature which is worth preserving. However, it would also be useful to be able to use a summarized form of a message to prevent fetching unwanted content (e.g., from recipients unknown to a user whose address is meant to be private). More generally, it would be useful to be able to authenticate senders (e.g., only accept messages from known senders, only accept messages from senders who can answer specific questions).

To summarize new requirements which arise out of this brief analysis:

1. composition – composing email with some knowledge of the recipient could reduce the need for unnecessarily sending it in multiple formats
2. attachments – a model for including documents in email
   (a) without unnecessary duplication

   (b) with reasonable copy semantics would be a useful improvement

   (c) supporting versioning for multiuser documents would also be useful

3. sending – support for unsending mail (completely if unread; informing the recipient it's been withdrawn otherwise) would be useful

4. authenticating and filtering senders – the sender should be able to choose from whom to receive messages

All of this functionality needs to be implemented in a secure framework. To do so, application of standard distributed systems design principles should be a good starting point.

3.2.2 Problems with Existing Model

The existing model runs into security problems primarily because it is not based on a notion of a secure distributed file store. When someone sends you an email, you are essentially giving them write access to your file system, which allows them to store a document (the email) plus the possibility of other documents (attachments) which may have unexpected semantics (running a program, executing a program contained in an attachment, running a script).

Privacy problems arise from the highly asynchronous nature of email: there need not be any connection at all between sender and recipient to initiate an email or to receive it. Consequently, it is not practical to authenticate senders. In the early days of the Internet when networks latencies were high, authentication was not an option. Even with generally lower latencies, authentication could pose a problem in the presence of temporary outages.

In what follows, a design is proposed which aims to solve both of these problems.

4 Distributed Systems Model

The idea explored here is to reconceptualize email (and potentially other services) as operations on a secure distributed file store. Sending an email corresponds to granting a (possibly) remote user permission to see a specific object on your part of the file system. Receiving an email corresponds to receiving this permission, and reading the email corresponds to using this permission to read the offered object.

The emphasis is taken away from messaging (a network-centric approach) to how information is stored transparently across a network (a distributed-systems approach).

For the model to be practical, receiving and reading an email would have to be implemented so that it became a local transaction – at least on a network closer than the sender, in the case where the two parties were very far apart.

A major benefit of going this route is that the receiver is no longer in effect giving possibly unknown outsiders write permission to their own file system. Handling attachments can also potentially be made more convenient.

The remainder of this section explores how this idea can be made concrete. First, suitable abstractions are outlined, followed by a discussion of how the distributed file store could be realised. An extention of the idea of distributed coordination is proposed, along with notification, to tie everything together. Finally, the key issues are summarized.

4.1 Abstractions

The abstraction most significant to differentiating the new approach is that of seeing email operations as operations on a distributed file store. Let's examine how the requirements of the new system are addressed by a distributed file store:

1. composition – there is the potential to query the recipient for allowable formats if the recipient is willing to make this information available; otherwise, a fallback position of either plain text or multiple formats is no worse than the existing model
2. attachments – addressed by including them as additional objects published to the recipient
   (a) duplication will only occur as needed for performance
   (b) the "sent" object could be a pointer or reference to the original (with read-only access), hence reflecting changes
   (c) versioning for multi-author documents could be supported with existing versioning tools, which becomes much easier when the "attachment" is a reference to the original, rather than a copy
3. sending – unsending becomes trivial; previously granted access rights can be revoked (though the practical effect of revocation assumes the message hasn't been read or copied)
4. authenticating and filtering senders – since the sender has to provide a capability with access rights to the object, the sender's identity could be encoded in the capability

The potential benefits appear to be sufficient to explore how the concept could be implemented practically.

4.2 File Store

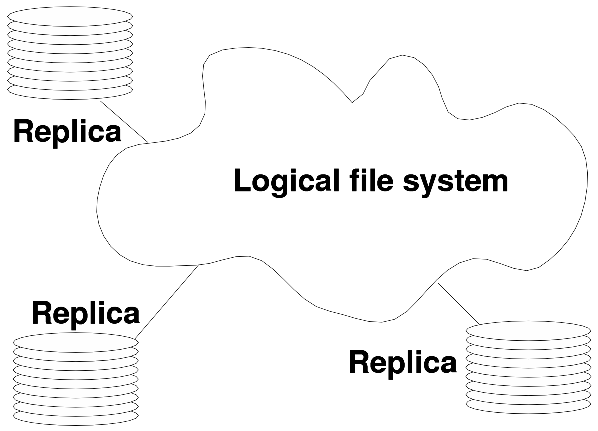

Figure 1: Distributed File System

Figure 1 illustrates the general idea of the distributed file store. Parts of the logical file store may be replicated at end points for efficiency, but it is logically a single file system.

For the file store idea to be practical, performance and security are critical issues to address. Since sending a message to someone implies that they will need read access to it, a push-based replication model, which, for example, has been shown to be efficient for web caching [Chen et al. 2003], is an obvious implementation strategy. While the recipient will be logically reading something in someone else's file store, it obviously needs to be cached locally for efficiency. Optimisations would be needed to cover cases like highly mobile users.

While implementation of such a scheme has potential for security holes, isolating the problem down to implementing a distributed, replicated file store reduces the range of potential problems. As compared with the existing email infrastructure, the problem is significantly reduced. In its existing form, email has the potential for security problems in eavesdropping, as well as contaminating the local file system of the recipient. Eavesdropping can be dealt with in the distributed file store by encrypting data, and using secure capabilities for enabling access.

Sending an email in the proposed model becomes a matter of sending a capability providing access to an object or objects in the file store to the recipient. The capability, in addition to the usual access rights, could contain a subject line, and the identity of the sender. The capability model should include features like encryption of capabilities, use-once capabilities, and revocable rights.

A file system such as Coda [Kistler and Satyanarayanan 1992], which has support for replication and disconnected operation, would be suitable as a basis for an initial implementation.

Figure 2 shows the sequence of steps in sending and receiving an email. The sender composes the mail, and sending it means sending a capability to the recipient. The recipient's nearest replica server may choose to replicate the original object before the recipient sees that there is an incoming message (which would look like the existing model), but this is not necessary. Only when the recipient actually wants to view the message would it be really necessary to replicate the content.

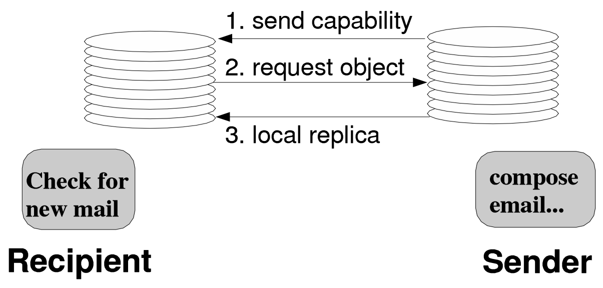

Figure 2: Basic Architecture

This basic model is general enough to support the abstractions needed for a range of message policies. For example, unsending a message can be accomplished by revoking a capability. Versioning in multiuser documents can be implemented by providing a version control system on top of the distributed file store, with capabilities issued to participants in creation or editing of a multiuser document.

4.3 Coordination and Notification

Coordination generally encompasses syncrhonization and communication [Tolksdorf and Glaubitz 2001]. However, in a distributed system over the entire Internet, the underlying idea of coordination – a separate abstraction for high-latency operations – can apply to other aspects of the design. Some examples

- entry to and exit from a multiuser document editing group
- managing locations of replicas
- managing distribution of security notifications

This idea is not, however, essential to the notion of re-conceptualizing the Internet as a distributed system. Coordination is introduced here to illustrate the range of ideas which could be drawn in to Internet services, from the general pool of distributed systems ideas.

A notification system such as that of Elvin [Sutton et al. 2001] could be used to inform users of new mail. Again, the detail is not critical to the model, but illustrates the range of options opened up by moving to a distributed-systems view of Internet mail.

4.4 Putting it All Together

Reconceptualizing the Internet as a platform based on a secure distributed file system opens up a range of possibilities for implementation, as well as having the potential to focus application-layer security problems into one subsystem.

The specific example of email illustrates the possibilities. Sending or unsending can be accomplished by giving out or revoking capabilites, with replication providing the mechanism for moving content closer to the recipient.

If email is split into sending a capability and retrieving the identified object, it becomes possible to send the recipient a specific document format (e.g., plain text, or HTML, depending on the recipient's preferences). The capability could point to all variants, and the recipient could choose which to receive.

A full implementation of the model would require mail clients to be replaced by software which used the distributed file store. As a phasing-in step, a web-based client could be provide access to the system for users without the new software. This approach to phasing the new model in would allow email to be replaced rapidly.

5 CONCLUSIONS

5.1 Summary

Implementing email via a replicated, distributed file store using capabilities to "send" messages is the major idea proposed here. The emphasis is on how to store messages and inform recipients of their existence, rather than on network protocols.

The proposed new architecture is not claimed to solve all security problems, but to provide a general abstraction for a range of Internet applications. While only mail is illustrated, the same basic mechanism could be used to publish documents of any kind. Instead of hyperlinks, capabilities could be made public, to create an equivalent effect to web pages, for example.

5.2 Way Ahead

The next obvious step is to implement a prototype of the proposed architecture as as proof of concept. However, it will not be a useful idea on its own without a roll-out model so that existing mail can be accommodated.

The next steps therefore are not only to implement a version of the proposal, but to investigate ways it can be integrated with conventional email, if only as a phasing-in measure. One option would be to have servers which could convert email to the new format. The message could be written into the shared space and the recipient notified, but only after the sender was asked to verify their identity. Even if this verification step was trivial, much spam would be eliminated.

Another option already outlined would be to have a web-based interface to the system. Attempts at emailing subscribers who have abandoned email could be redirected to the web-based front end (which could include information on how to start using the new system).

More detail of phasing in models needs to be considered to make the scheme viable as, despite its flaws, email is too pervasive to replace overnight.

5.3 Overall Conclusions

Spam, worms, denial of service attacks and malicious attachments are just some of the problems with email in its current form.

The proposed model may not solve all these problems, but distributed systems ideas which have evolved over several decades are worth exploring as an alternative. Whether the model outlined here is the best starting point or not is not the big issue: the important issue is to evolve designers' thinking away from a networked mindset to a distributed systems mindset. If this can be achieved, several decades of research can be used as a basis for modernising Internet applications.

References

Aberer, K. and Despotovic, Z. (2001). Managing trust in a peer-2-peer information system. In *Proceedings of the tenth international conference on Information and knowledge management*, pages 310–317, Atlanta, GA. ACM Press.

Bellovin, S., Schiller, J., and Kaufman, C. (2003). RFC 3631:security mechanisms for the internet. WWW Document.

Chen, Y., Qiu, L., Chen, W., Nguyen, L., and Katz, R. (2003). Efficient and adaptive web replication using content clustering. *IEEE Journal on Selected Areas in Communications*, 21(6):979–994.

Kistler, J. J. and Satyanarayanan, M. (1992). Disconnected operation in the Coda file system. *ACM Trans. Comput. Syst.*, 10(1):3–25.

Landwehr, C. E., Bull, A. R., McDermott, J. P., and Choi, W. S. (1994). A taxonomy of computer program security flaws. *ACM Comput. Surv.*, 26(3):211–254.

Park, Y.-J. and Lee, G. (2004). Repairing return address stack for buffer overflow protection. In *Proceedings of the first conference on computing frontiers on Computing frontiers*, pages 335–342. ACM Press.

Pinkas, B. and Sander, T. (2002). Securing passwords against dictionary attacks. In *Proceedings of the 9th ACM conference on Computer and communications security*, pages 161–170. ACM Press.

Shirey, R. (2000). RFC 2828: Internet security glossary. WWW Document.

Sipior, J. C., Ward, B. T., and Bonner, P. G. (2004). Should spam be on the menu? *Commun. ACM*, 47(6):59–63.

Sirbu, M. A. and Chuang, J. C.-I. (1997). Distributed authentication in kerberos using public key cryptography. In *Internet Society 1997 Symposium on Network and Distributed System Security*. `http://www.ini.cmu.edu/netbill/pubs/pkda.pdf`.

Sutton, P., Arkins, R., and Segall, B. (2001). Supporting disconnectedness – transparent information delivery for mobile and invisible computing. In *Proc. 1st International Symposium on Cluster Computing and the Grid*, pages 277–285, Brisbane, Australia.

Tanenbaum, A. S., van Renesse, R., van Staveren, H., Sharp, G. J., and Mullender, S. J. (1990). Experiences with the amoeba distributed operating system. *Commun. ACM*, 33(12):46–63.

Tanenbaum, A. S. and van Steen, M. (2002). *Distributed Systems: Principles and Paradigms*. Prentice-Hall, Upper Saddle River, NJ.

Tolksdorf, R. and Glaubitz, D. (2001). XMLSpaces for coordination in web-based systems. In *Proc. Tenth IEEE Int. Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pages 322–327, Cambridge, MA.

Weaver, N., Paxson, V., Staniford, S., and Cunningham, R. (2003). A taxonomy of computer worms. In *Proceedings of the 2003 ACM workshop on Rapid Malcode*, pages 11–18. ACM Press.

Zouand, C. C., Gong, W., and Towsley, D. (2004). *Feedback Email Worm Defense System for Enterprise Networks*. Technical Report TR-04-CSE-05, University of Massachusetts, Amherst.