ERIC ALLMAN, SENDMAIL

E-mail Authentication

what, why, how?

Internet e-mail was conceived in a different world than we live in today. It was a small, tightly knit community, and we didn't really have to worry too much about miscreants. Generally, if someone did something wrong, the problem could be dealt with through social means; "shunning" is very effective in small communities.

Perhaps we should have figured out what was going to happen when Usenet started to go bad. Usenet (aka Netnews) was based on an inexpensive network called UUCP (quaintly standing for Unix to Unix copy program), which was fairly easy to join, so it gave us a taste of what happens when the community becomes larger and more distributed—and harder to manage. Even the worst flame wars seemed fairly innocuous in the grand scheme of things, and kill files were really enough, but there was a seed of something ominous that was going to germinate all too soon.

Although the Internet went live in 1983, the real explosion occurred in the early 1990s, when the number of Internet hosts grew from under 100,000 in 1989 to around 5 million by the middle of the decade,[1] rising to nearly 40 million by early 1998. That was the year that Hotmail (launched in mid-1996) was sold to Microsoft for a reported $400 million,[2] with around 30 million users.[3] Coincidentally, the latest surveys indicate about 400 million Internet hosts as of the beginning of 2006,[4] and Microsoft claims 261 million Hotmail accounts as of June 30, 2006.[5] A lot of people have discovered the Internet,

E-mail Authentication
what, why, how?

including a lot of consumers. Of course, businesses follow consumers, so much so that URLs are ubiquitous today on everything from the sides of buses to water bottles to toothpaste tubes. Businesses call consumers "customers," and they generally try to extract money from them, which brought money into the picture. And money attracts miscreants: everyone from small-time hustlers to organized crime members.

There are really two major communication technologies on the Internet today: the Web (a pull technology), and e-mail (a push technology). There are many others, of course, such as pub/sub services, IM, VoIP, etc., but today these two remain the granddaddies of the technology. In theory pull technologies should be more secure, since the user has to go out of his or her way to access the site, but in fact that is no more than a tiny speed-bump on the information superhighway. That's not the theme of this article, however.

E-mail has had a fundamental flaw from the beginning: a lack of authentication. This means that anyone on the Internet can, in theory, send e-mail to anyone else while claiming to be a third person. The reason for this is purely historic: E-mail was one of the very first network protocols—in fact, the first three application layer protocols published for the Internet were telnet, FTP, and SMTP[6]—even before DNS, and before there was any infrastructure to support distributed authentication in the nascent Internet (telnet and FTP used local authentication). Coupled with greed, this is a dangerous property. For example, I have no way to prove that a message that claims to be from my bank actually has anything to do with my bank. This situation is increasingly untenable, which is resulting in calls for authentication technology.

With literally hundreds of millions of mail clients and servers out there, updating the mail protocols cannot be done lightly. A change that would break existing mail clients or servers would be disastrous. Such a change would also take a long time. We made a major change in SMTP in 1993 to support extensions such as eight-bit data and encryption (ESMTP, RFC 1425, later updated), and there are *still* a few servers that have not been updated well over a decade later. We should expect a similar adoption cycle for authentication and plan for a system that can be deployed over a long period without breaking existing servers and creating "walled gardens" that can talk only among themselves (that is, systems with authentication must interoperate with older systems).

Another issue is that we still don't have a globally accepted distributed authentication infrastructure. Many proposals have been put forth, but none has become ubiquitous. There are proposals for e-mail authentication, however, that avoid this need.

CURRENT PROPOSALS

Authentication proposals come in two flavors: path-based and signature-based. There are many variations on this theme.

Path-based algorithms look at where the mail was sent from. For example, if I receive an e-mail from a site claiming to be mybank.com, I want to see if the client sending the mail is actually owned by MyBank. I can determine this by asking mybank.com what hosts send mail on its behalf. Since I'm going to the real mybank.com to get the information, I should be able to trust that data. If the message comes from an address that MyBank does not send mail from, then the message is probably suspect.

This gets a bit complex, however, because of the way business is done today. For example, MyBank might well hire a company to send e-mail on its behalf, using a MyBank e-mail address as the sender. This is common for marketing, human resources (e.g., payroll, benefits), front-line support, CRM, etc. This can be dealt with, albeit painfully.

The real difficulty is that the Internet e-mail model allows messages to be forwarded. For example, professional and alumni organizations commonly have e-mail forwarding services that let you have a single address that does not change even when you change jobs or ISPs. I might have a forwarding address at alumni.myuniversity.edu that sends the message to my ISP, allowing me to move from one provider to another without having to change my e-mail address. Thus, when I receive the message, it is going to seem to come from my forwarding service, regardless of who actually sent it. In some cases these problems can be dealt with, but not without difficulty.

The best-known examples of path-based authentication are SPF (Sender Policy Framework) and Sender ID.

Signature-based algorithms, on the other hand, determine whether the message is legitimate by using a cryptographic digital signature on the message. Public-key encryption allows the signer (usually the sender of the message) to publish its public key so that the verifier (usually the recipient) can verify that the message is properly signed. For example, if I get a message that claims to be

signed by MyBank, I can ask mybank.com for its public key and then use that key to verify that the signer really was MyBank and not someone else spoofing the message.

The third-party (outsourcing) problem can be handled in several ways; the simplest way is to give the outsourcer a key that it can use to sign on behalf of the claimed signer. Of course, this needs to be protected so that the key can't be used to sign a message claiming to be from the CEO, but this is relatively easy as well. There is no problem with forwarding messages, as long as the forwarder doesn't modify the message in the process (which would break the signature). The biggest problem is likely to be with mailing lists that do modify the message (for example, by adding unsubscribe information); such list management software has to be updated.

The best-known examples of signature-based algorithms are DomainKeys from Yahoo! and DKIM (DomainKeys Identified Mail), the result of a group effort that started with DomainKeys and IIM (Identified Internet Mail) from Cisco. These are similar algorithms, but they do have distinct differences.

SO WHICH ONE IS BEST FOR ME?

Naturally, there are trade-offs among these various technologies. Path-based systems are very simple for senders: They just install a new record into DNS. Receivers have to install some new software, so it's a little harder for them. In contrast, signature-based systems require software at both ends, but they aren't confused by message forwarding and are somewhat more flexible for senders. For example, if one site hosts more than one domain (that is, two or more domains can send from the same IP address), then any of those domains can masquerade as any of the other domains in path-based systems. In signature-based systems each can have its own key.

Many sites are already using both schemes and open source implementations of all the options. The most widely deployed is sender-side Sender ID/SPF, probably owing to the simplicity of creating the record (unfortunately, it's nearly impossible to determine how many sites are checking this information). Signature-based algorithms, however, are already being used by many big hitters, including ISPs and large enterprises, and they are supported by most providers of anti-spam technologies.

For the best protection you should probably use at least one of each type of system. Sender ID and SPF are close enough that you don't have to choose between them. At the moment, DomainKeys is more broadly implemented than DKIM, but in the longer term DKIM is likely to be preferred because of additional functionality and security as well as the expectation that it will be an official Internet standard. DomainKeys and DKIM can coexist, so it's quite feasible to install both of them at the same time.

WHAT ABOUT S/MIME OR PGP?

Some of you may be wondering why we aren't just using S/MIME (secure MIME) or PGP (Pretty Good Privacy), existing e-mail security technologies that provide encryption and signing. The simple answer is that they do the wrong thing for our needs. This is not to say they aren't valuable; they are, and they should be more broadly deployed. Both of them, however, are victims of the key-management problem, and both are intended for user-to-user rather than server-to-server use, which limits their use for server-level filtering.

Key management comes from the way that the verifier ensures that it has the correct public key for the signer. Schemes such as DKIM just ask the signer for its key using DNS, very similar to the way they ask a domain for its IP address when sending mail. S/MIME and PGP, however, both use schemes where another entity vouches for the key by having that third party sign that key. Of course, to verify that signature, the key of the signer needs to be signed. S/MIME uses these key signatures to arrange all the keys into a tree (really several trees); the roots of those trees are well-known (and carefully guarded) entities, usually companies, called certificate authorities. The verifier (such as your Web browser) has a list of well-known certificate authorities built in, which is how secure Web sites prove who they are. PGP uses a "web of trust": anyone can sign anyone else's key, and you follow your way through that web until you find someone you trust. Hierarchical structures are not required, and certificate authorities are not necessary.

Asking everyone who wants to send mail to get a signed key is difficult (and probably expensive); systems such as DKIM allow anyone to create his or her own key and publish it without going to a third party.

Another reason not to use S/MIME or PGP is that it changes the body of the message in a way that is likely to be visible to the message reader, particularly if the reader isn't using the same authentication scheme. One of the goals of DKIM was that receivers that didn't implement DKIM wouldn't see any change in what is displayed to the end user.

WHAT NEXT?

One of the classic misconceptions is that authenticated messages can be trusted. By the very nature of authentica-

E-mail Authentication
what, why, how?

tion, spammers and phishers can authenticate themselves as well as legitimate senders can. Once you know who someone really is, you also need to know whether this is someone from whom you want to accept mail. The major schemes for this are accreditation and reputation.

Accreditation is based on third parties that audit senders to ensure that they follow good practices (such as using only double-opt-in lists and immediately honoring all unsubscribe requests). The accreditor then makes it publicly known that the sender is legitimate. Generally, accreditors will then monitor the behavior of the senders to verify that they are following the approved practices. Accreditors must themselves be held in high regard, so they have a strong incentive to make sure that the people they accredit obey the rules.

Reputation generally also involves third parties that monitor how e-mail senders behave. This monitoring can in most cases be done without the cooperation of the various senders on the network. For example, they set up honey pots to attract spam, monitor abuse complaints, and collect community feedback. Sites that are known to send a lot of spam get poor reputations, whereas sites that send a lot of e-mail but have few complaints get good reputations. They can then publish this information to receivers. Large sites, especially large ISPs, can collect this information internally and make exceptionally good decisions. Smaller sites will probably have to subscribe to a service that amalgamates data.

There are also more immediate ways to use authentication data. For example, as defined, none of these schemes specifies a way that an end user will see authentication information. An e-mail provider might want to warn me, however, if I receive a message that claims to be from MyBank but is actually from elsewhere. This alone would give the phishers a serious case of heartburn. Users could have their own personal allow-lists, probably automatically generated by scanning their address books and monitoring whom they communicate with.

Using authentication information effectively will be the next big challenge. It will impact e-mail systems on many levels: clients, servers, and additional tools. Some early work is available on this topic, with more to come.

E-mail authentication is definitely coming, and, in fact, is arguably already here. The next big step is figuring out how best to use that information. Authentication also enables a whole new class of anti-spam and anti-phishing algorithms. There is still much to learn. Q

REFERENCES

1. Salus, P. H. 1995. *Casting the Net*. Boston: Addison-Wesley Professional.
2. Pelline, J. 1998. Microsoft buys Hotmail. CNET News.com; http://news.com.com/2100-1033-206717.html.
3. Microsoft. 1999. MSN Hotmail: From zero to 30 million members in 30 months; http://www.microsoft.com/presspass/features/1999/02-08hotmail.mspx.
4. Internet Systems Consortium. ISC Internet Domain Survey; http://www.isc.org/index.pl?/ops/ds/.
5. Microsoft. 2006. Microsoft reports fourth quarter results and announces share repurchase program; http://www.microsoft.com/msft/earnings/FY06/earn_rel_q4_06.mspx.
6. Defense Communications Agency. 1985. *DDN Protocol Handbook*, Volume 1, DOD Military Standard Protocols. NIC 50004 (December).

MORE INFORMATION

http://dkim.org
http://antispam/yahoo.com/domainkeys
http://openspf.org
http://microsoft.com/senderid

LOVE IT, HATE IT? LET US KNOW

feedback@acmqueue.com or www.acmqueue.com/forums

ERIC ALLMAN is the cofounder and chief science officer of Sendmail, one of the first open source-based companies. He was previously the lead programmer on the Mammoth Project at the University of California at Berkeley. This was his second incarnation at Berkeley, as he was the chief programmer on the INGRES database management project. In addition to his assigned tasks, he got involved with the early Unix effort at Berkeley. His first experiences with Unix were with 4th Edition. Over the years, he wrote a number of utilities that appeared with various releases of BSD, including the -me macros, tset, trek, syslog, vacation, and, of course, sendmail. He spent the years between the two Berkeley incarnations at Britton Lee (later Sharebase) doing database user and application interfaces, and at the International Computer Science Institute, contributing to the Ring Array Processor project for neural-net-based speech recognition. He also coauthored the "C Advisor" column for *Unix Review* for several years. He was a member of the board of directors of Usenix Association.