At the end of a session, the filesystem is characterized by performing a recursive checksum on all files and checksumming the directories. The checksum of a directory can be used to tell whether the tree under that directory needs to be refreshed. The checksums are generated by the checksum tool and used by the comparison tool either on a file by file basis or in a batch mode.

Because the system was developed to employ only user level tools (the only administrative changes were to allow NFS export of the home area on the work system), some pieces of state live in the filesystem as regular files. The checksum files and files to act as deleted file markers are distinguished by naming convention only, being "invisible" to the user by virtue of being dot files. Combined with the need not to modify the system, these features may have caused the system to be described as a hack during the questions, but Dan does have a system that allows him to work that he constructed with user-level tools.

Invited Talks

If Cryptography is So Great, Why Isn't it Used More?

Matt Blaze, AT&T Labs

Summary by Gordon Galligher
<gorpong@psa.pencom.com>

This talk was based much more on the technical reasons for not using cryptography than on the various political reasons that abound, especially in the US. Matt spent some time (ten minutes) covering the lowest level of information one needs to know about cryptographic technology to understand the rest of the talk. These are summarized below:

Message Authentication Code (MAC) Functions provide integrity checking and authentication based on a shared key to compute a code for the message. This has been all there was for the past 3,900 years. Only in the past 100 years have mathematicians been concerned with cryptography.

Secret-key Cryptography requires shared secret of the encryption key. There are block ciphers, which encrypt fixed-size blocks, and stream ciphers, which encrypt using a sequence of "mask" bits.

Public Key Cryptosystems are based on two keys, one for encryption and one for decryption. It is also based on the assumption that one cannot calculate the private key based on the public key only. This is a conceptually simple concept, but it has some annoying details. Most systems use large numbers (1,000 bits), so the encryption is very slow and care must be taken for small messages and the actual key generation.

Digital Signature Schemes are similar to a MAC, but have two keys. They establish integrity and authenticity, but also allow third parties to verify the signature (i.e., a receiver cannot forge anything).

Secure Hash Functions reduce a message to a fixed size, then compute a signature. They are one-way operations and are generally used as "fingerprints" for a particular message.

Matt then went right into his top ten cryptography problems. First, you never needed cryptography before, but you do now. "Important stuff" was typically handled in person, by hand via contracts, letters, messengers, etc. Now it is done electronically, and the best (i.e., cheapest, fastest) medium is *less* secure and will get worse. The Internet and cellular phones are prime examples.

Second, no one realizes this necessity. Everyone agrees that security is a great thing, but no one wants to pay for it. There is a slight cost investment, but cryptotechnology is fairly inexpensive. There is decreased performance, but this is also not a major problem with faster and faster hardware. The largest problem is that it is harder to use! To truly be effective, it must be transparent. As an example of this, he asked the audience if they thought they could get their boss to use PGP– the moral being that it would be too difficult and the boss would just not do it.

Third, you cannot really put cryptography where you need it. There are problems with performance vs. security vs. functionality. There are some engineering issues such as caching, memory (the plain text can still be seen in memory), and compression. Using compression really screws up cryptography, plus if something is encrypted, it is damaging to the compression algorithm. Security often favors moving cryptography close to the application to increase control, but usability concerns usually end up pushing the cryptography away from the application.

Fourth, secret keys often aren't. There are some implicit assumptions made in these systems: the key space is too large to make an exhaustive search for the correct key a practical alternative; the key is truly random; and the key is kept from the "bad guys." Furthermore, weaknesses in this aspect are extremely common.

The technology to search various keys typically improves exponentially because historically the performance:cost ratio of technology doubles approximately every 18 months (Moore's Law). A recent study using this has shown that a 90-bit key should be sufficient as a secret key for the next 20 years. Unfortunately, the Data Encryption Standard (DES) uses a key that is only 56 bits long, with the exportable version limited to 40 bits. This is clearly less than 90 bits, and Matt mentioned that this is "woefully insecure."

Fifth, public key infrastructure does not exist yet. In theory, the use of public key algorithms eliminates the need for a pre-agreement between people who want to share information. The fact is that if there is no central authority, it is difficult to find the person's public key; and once we find a key, how do we know it is the "real" one? One answer is to use a certificate authority to "bind" a particular key to its owner. This authority must, however, be very well known with a very well known key. This requires a certificate infrastructure that is just not available today. This does not even begin to touch the situation that certification is *not* the same as trust. "Trust management" is an open research area.

Sixth, cipher algorithms are amazingly hard to design. Most of the secret key ciphers are just ad hoc. Many times they are designed specifically to resist known attacks, which may or may not have them susceptible to new attacks. Public key ciphers attempt to reduce some of the "harder" problems, such as factoring; but the proofs of this "secureness" are hard to come by. The world has remained relatively safe only because most people do not know how to create these algorithms.

Seventh, cryptography protocols are really, really hard to design. The algorithms are merely the building blocks to create the actual protocols. It is incredibly hard, but it looks quite easy, so people try to design their own constantly. This, by the way, is one of the things that keeps the "bad guys" in business.

Eighth, designing a secure application is even harder. The protocol is merely a mathematical abstraction, whereas the designer of the application must pick the most appropriate protocol and must figure out the assumptions in the protocol and be sure to meet them. Even simple examples are difficult because no one really knows if *any* crypto application is truly secure in practice. All we know is that no one has come forward with information on how it was broken. This comment really chilled me. It is not even "no one has broken it," but simply "no one has *claimed* to have broken it," and we really do not know.

Ninth, cryptography implementation is hard as well. No one really knows how to write correct software. There are always things such as data type mismatch errors, endian errors, especially when cross-platform development occurs. In some cases important steps of the protocol are left out, possibly resulting in sending cleartext instead of ciphertext. Matt mentioned that there was a fairly popular implementation that made a subroutine call and did *not* check the return code. That subroutine was to set the cipher key to use for the ensuing communication. There was a certain set of situations that could cause a *null* key to be generated, which is the same as *no* key. Because the return code was not checked, the program had no way of knowing that there was no encryption being performed. Furthermore, generic security bugs do not cause obvious failures in the application.

Tenth, cryptography does not magically make insecure platforms secure. Examples of standard security problems include network breakins, viruses, sendmail, users, etc. Even most of the "secure" platforms are not truly secure, because they may have the information left in their memory cache or paging areas. The easiest attacks to use typically do not attack the areas that are easiest to protect with cryptography. This means that cryptography really does not address most of the security threats that exist today.

Problem ten was not the last one. There are many other technical problems, such as security bugs not behaving like other bugs and the crucial one that no one is allowed to make a mistake when designing/running applications! There are also a number of social, political, and economic issues with introducing encryption technology to regular use. As an example of this, Matt asked who bought the *first* FAX machine. When this person bought it, to whom were the FAXes sent? There are also "crazy government regulations" that keep cryptographic techniques classified as a munition, thus limiting the export possibilities of the technology. When faced with trying to get around the crazy hoops of the US Government to export encryption technology, many companies just give up and do not include encryption technology in their products.

Matt further mentioned that what cryptography does is really close to what we need, in terms of security, and that it is very tempting to pretend that it does solve the problem. He closed with a very appropriate quote: "in theory, theory and practice are the same, but in practice, they're different." Nowhere is it truer than in cryptography.

The Inktomi Search Engine

Eric Brewer, University of California, Berkeley
Summary by Jerry Peek
<jpeek@jpeek.com>

Inktomi is one of several popular search engines on the World Wide Web. Like other search engines, Inktomi lets Web users hunt for Web pages that contain a word or phrase. The world of search engines is competitive, and Eric Brewer didn't miss many chances to say where Inktomi is superior to other search engines, especially to AltaVista. His talk also had plenty of technical information, including some about AltaVista that wasn't covered in the AltaVista session. [Also see the next session report.]

In general, search engines have three parts. A *crawler* searches the Web for pages (documents) by following hypertext links from other pages. A *database* stores information about the pages. And an *HTTP interface* (basically, a Web server) handles users' queries on the database.