TRUFFLES — A SECURE SERVICE FOR WIDESPREAD FILE SHARING

Peter Reiher
Thomas Page, Jr.
Gerald Popek

UCLA
Los Angeles, CA

Jeff Cook

Trusted Information Systems
Los Angeles, CA

Stephen Crocker

Trusted Information Systems
Glenwood, MD

ABSTRACT

Truffles is a system meant to address some of the major issues that still make it difficult to share files between users at different sites. In particular, it addresses the problems associated with secure file sharing, and the problems of high administrative overhead. Truffles will combine facilities of the Ficus file system and TIS/PEM, a privacy enhanced mail system, to make file sharing considerably easier. Truffles must deal with several important security problems, including secure transport of data, authentication of the users sharing files, handling of different administrative domains, and permitting system administrators to control, flexibly, yet easily, what sorts of sharing are done. This paper describes these problems and the solutions Truffles will use.

INTRODUCTION

Users who share a single machine, or who share a single administrative domain over a local area network, are able to share files with each other very easily. Users can share source code for programs they are developing, work together on papers and other documents, and use common shared libraries and programs. Users who are not so closely connected physically currently cannot share files with the same ease. Although the complicated nature of the networks connecting such users used to be the primary impediment to sharing, today the lack of a common administrative domain and security concerns are the major unsolved problems.

Truffles (TRUsted Ficus FiLE System) is a system that attempts to make file sharing between users in different domains both simple and secure. The motivation for Truffles is the question: Why is it harder to share a file with an arbitrary remote user than it is to send him electronic mail? Sending electronic mail to another user is a relatively simple operation in today's networks. The sender need merely know the address of the receiver and have reasonable network connectivity. By invoking a single program, giving it only the receiver's address and the message, the sender can be reasonably certain that his mail will reach its destination, if possible. The sender need not worry about the path it takes, whether machines along that path fail, the hardware types involved, or any of the other complexities of the worldwide network of computers. It is really no harder than sending electronic mail to a user on the same machine, despite the fact that the operations that have to be performed to deliver the mail are much more complicated.

Sharing files between users on different machines, on the other hand, is not nearly as easy as sharing them with a local user. One can use electronic mail to ship text files back and forth, but many electronic mail systems do not handle non-text files well, propagating changes in the files must be done by hand, electronic mail will not coordinate access to the files, and, unless a secure form of mail is used, electronic mail offers little protection.

Some tools more specialized for file handling exist, such as ftp, telnet, and NFS [1]. Generally, however, these tools have significant disadvantages. None of them handle security issues well. Telnet and ftp also do not use the same user interface that normal local operations use, and do not easily permit users to mix local and remote operations. NFS and other network file system services do a better job of unifying the local and remote cases, but they require substantial setup service by the participating sites' system administrators, and they generally have poor availability in the face of failures. Further, since all operations have to fetch their data across the network, these systems can be slow and/or expensive.

The Ficus file system has solved the problems of poor availability and performance by replicating files for users [2]. Since users at different sites have local versions of their shared files, they are not as affected by failures, and they get substantially the performance of local access. But Ficus only works well in shared administrative domains, and does nothing to address the security concerns.

Truffles is an attempt to solve the problem of permitting controlled file sharing between users in different administrative domains. Only those users and sites that are permitted to participate should be able to do so. Those who are not permitted to participate should not be able to eavesdrop on data belonging to the relationship, nor should their requests for data or updates be honored. Another important security aspect of the problem is that the sharing should be limited strictly to what the participants intend to share. A participant who only meant to share a few files should not be forced to grant access to his entire machine.

A file sharing service of this kind cannot be successful if it relies on constant system administrator intervention. Just as users do not typically ask their administrator if they can change the access permissions on their files to make them available to other local users, they should not necessarily have to consult their administrator before sharing them with remote users. If setting up every sharing relationship with an outside user requires positive action on the part of the system administrators of the machines involved, few relationships will be set up. On the other hand, system administrators must be able to exert some form of control on what their users can make available to the outside world.

Finally, Truffles must make the sharing easy and painless, both in the initialization phase and during ongoing operations. The operations to establish the relationships should be simple and straightforward. Truffles should require little or no human intervention to keep a sharing relationship going. Only when the relationship is being changed, such as adding or dropping participants, should any of the users be reminded that the shared files are any different than any of their other files. Even then, the reminder should be painless and easy to deal with.

This paper describes the Truffles approach to solving this problem, with particular emphasis on the security issues. The next section describes the general approach. The following two sections describe two major software components that will be used in building Truffles. The section after those describes the overall Truffles architecture, particularly details touching on security. The following section briefly surveys some related work. The final section describes the current state of Truffles, its future, and gives some conclusions.

THE TRUFFLES APPROACH

The Truffles approach to this file sharing problem is to provide a secure file sharing service usable over normal network connections. Setup will be done through an electronic mail interface. File sharing has normal Unix file semantics, once the relationship has been established. Because the connections between sites using Truffles may have high delay, and the networks or sites may fail, Truffles automatically supports keeping multiple copies of a file on different machines.

Truffles is meant to run in a UNIX environment, on sites running standard UNIX operating systems with minor modifications and some additional software. The security goals of the Truffles system are not to improve the existing security features of UNIX systems, but to extend the existing level of UNIX security to files that are shared across administrative boundaries over insecure networks.

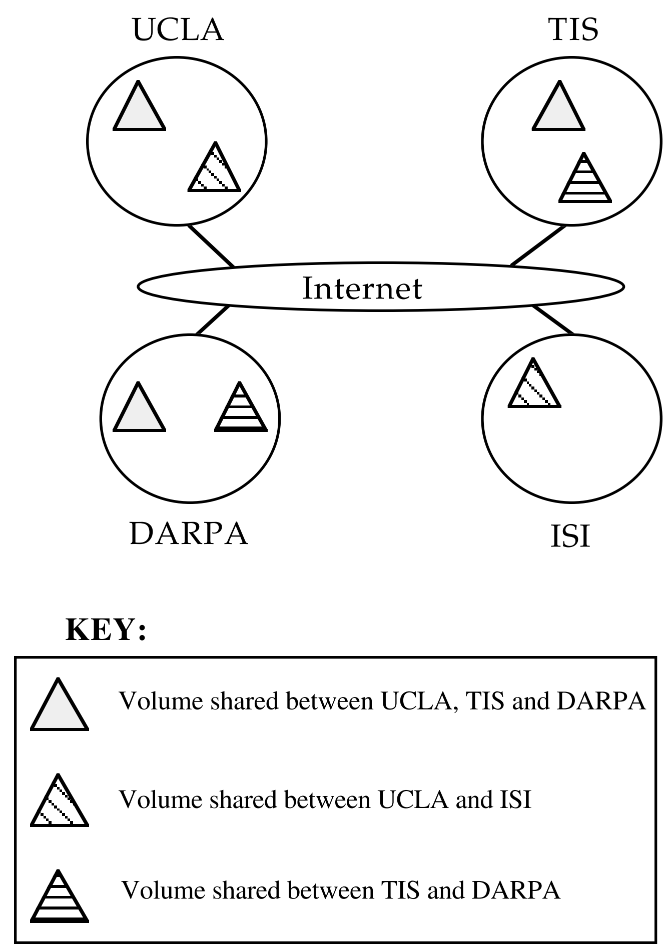

Figure 1. Sites sharing volumes via Truffles

Truffles file sharing

Truffles can provide file sharing either through replication or transparent remote access. Truffles provides file replication on a per-volume basis, rather than a per-file basis. A *volume* is similar to the concept of a Unix file system. It consists of a connected tree-like structure of directories and files, all stored on a single physical device. Any files that are to be replicated must be collected into a volume or set of volumes. Any files in those volumes will be shared, with normal Unix permission controls dealing with access to them. Truffles permits multiple sites to store replicas of a volume, and other sites to participate without storing a replica (at the cost of some performance degradation and inferior availability when sites fail).

Figure 1 shows how several sites might share Truffles volumes among themselves. In this example, four sites (UCLA, TIS, DARPA, and ISI) share three volumes (represented as triangles) among themselves, using the Internet to provide transport services. UCLA shares a volume with DARPA and TIS, TIS and DARPA share another volume, and UCLA and ISI share a volume. Different users might have set up and used each relationship. Each relationship is separate from the others, and does not depend upon them. Depending on how permissions are set and name spaces are organized, users at the four sites might or might not be able to access files whose volumes are not locally replicated.

Figure 2 shows another view of how Truffles volume sharing works. This figure shows the top levels of the file hierarchies on two sites, UCLA and TIS. Part of each site's file hierarchy contains files that are stored with normal UNIX file systems, such as the files under `/usr` and `/etc`. Another part of each site's hierarchy contains Truffles files. Only files in the Truffles parts of the hierarchies can be shared. Files in the non-Truffles parts of the hierarchies are completely inaccessible via Truffles. This design has an important security implication — any files not stored in the Truffles part of a site's namespace are completely shut off from sharing via Truffles, and are inaccessible to remote sites using Truffles.

Within the Truffles portion of the file hierarchy shown in figure 2, files are organized into volumes. The triangular shaded areas of the hierarchy show volume delimitations. The two sites do not necessarily share a common namespace. In this example, the root of the UCLA Truffles file system is called `/global`, while the root of the TIS Truffles file system is called `/g`. Some portions of the namespace are shared, though. The Truffles volume rooted at `/global/us/edu/ucla/reiher/shared` in the UCLA hierarchy has a replica at `/g/us/com/tis/cook/shared`. Despite the two replicas being stored at different places in the hierarchies, Truffles will keep all files in the two replicas consistent. Alternately, sites are permitted to completely share identical Truffles namespaces.

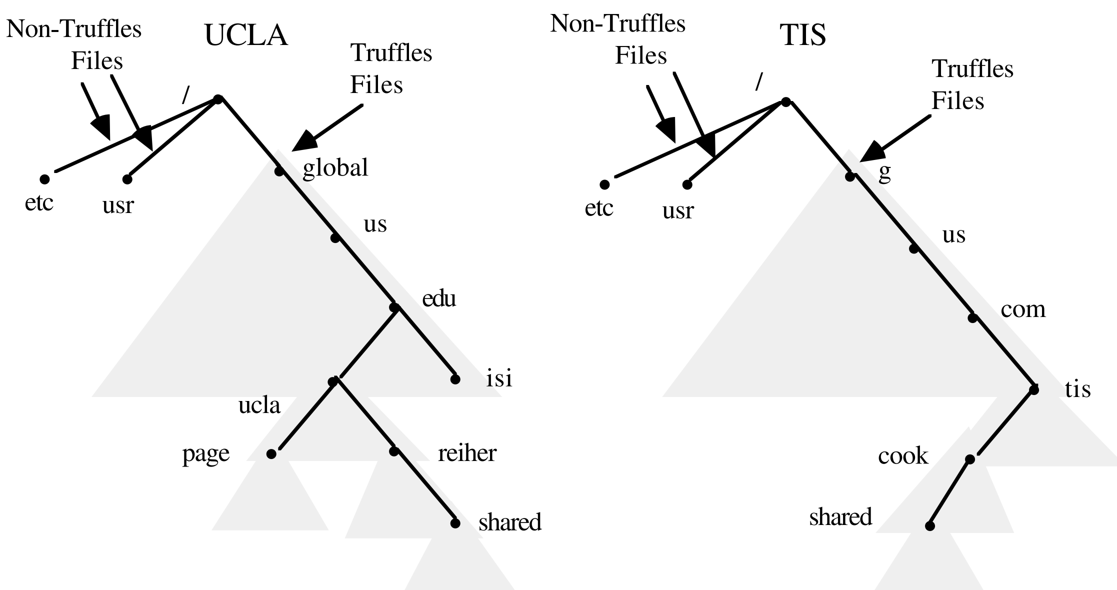

Figure 2. File hierarchies using Truffles

If another machine at TIS stored all of the TIS-local Truffles namespace up to, but not including, the volume shared between TIS and UCLA, users on that machine would be able to use Truffles to remotely access the shared volume, to the extent that normal access permission mechanisms allowed. Truffles provides not only replication services, but transparent remote access. However, the portion of the namespace not shared between TIS and UCLA is not accessible to remote sites through Truffles. For example, users at TIS are not able to see the volume `/global/us/edu/ucla/page`, and cannot use Truffles to get at that volume in any way, short of setting up an explicit relationship to share that volume. In fact, if the single shared volume shown in the diagram were the only volume shared between TIS and UCLA, only those files would be jointly accessible by both sites. UCLA could not use Truffles to examine any other TIS files, and TIS could not use Truffles to examine any other UCLA files.

Constructing Truffles

Truffles is being built from two existing pieces of software. The Ficus file system will provide file sharing and replication. The TIS implementation of Privacy Enhanced Mail (TIS/PEM) will provide a secure channel for the setup traffic and distributes the keys used for authentication and encryption. The full Truffles system will require a merging of these two components, with minor modifications, plus a reasonable amount of additional software. The components not directly provided by Ficus or TIS/PEM include

- the protocol for setting up a relationship
- daemons to handle most of the setup work without user intervention
- secure transport of data in an established relationship
- handling user identifiers between different administrative domains
- mechanisms and policies to control file sharing

Broadly, the Truffles approach is to use TIS/PEM, which can both authenticate and encrypt electronic mail, to send the messages between users to determine that a relationship is desired, and to authenticate each other. The use of electronic mail to establish the connection has certain advantages over other alternatives. The cooperating users need only know each other's electronic mail addresses. There is no need to request direct intervention of a system administrator to set up the connection via some other mechanism, like NFS. Electronic mail is able to handle issues like temporary failure of the destination site gracefully.

TIS/PEM is also used to establish what encryption keys will be used for this sharing relationship. Truffles daemons then take over the rest of the protocol to set up a shared set of files. This protocol consists of trading electronic mail messages between daemons that run Ficus utility programs in response to the messages. Since these utility programs permit remote users to gain access to local files, all messages in the protocol must be authenticated.

Once the relationship is established, all users involved in it will see the volume at the appropriate places in their file hierarchies. Ficus will ensure that all updates are seen by all replicas. Ficus will also deal with any problems arising from failures, recoveries, and partitions. From the users' points of view, the situation is little different than if they shared the files with each other on a single machine.

Truffles design problems

Truffles must deal with some other problems related to establishing and maintaining the sharing relationship. First, there are policy questions concerning which users can share which files with which sites. The answers to such questions are likely to vary greatly from site to site, depending on the importance of the data, the trust in the users, and the caution of the system administrator, among other factors. Therefore, Truffles will provide a mechanism for validating each request to set up a file sharing relationship, but will keep the policy well separated from the mechanism. Depending on circumstances, relationships might be permitted without any checking, or only between approved sites and users, or only for certain volumes, or only if the system administrator has previously granted permission, or only if the system administrator actually reviews and approves the request. The mechanism will be sufficiently flexible to permit these, and many other, policies to govern sharing. Policy will be under the control of local system administrators.

Another problem is that different sites have different low level names for their users. In the Unix world, each user on a site has a unique identification number, called a UID. But this UID is unique only to that site, or to that site and others sharing the same name assignments. In general, this UID might be used for a different user on different sites. But Truffles must not permit one of those other users to be mistaken for the local user, simply because they share a UID. Truffles must be careful to map remote users' UIDs without mistakes.

Truffles must also provide secure transport of the data. Ficus itself does not depend on secure data transport, as it was originally built to run on a trusted network. In the world of the Internet, however, Truffles messages could easily be read by eavesdroppers, or improper messages could be injected into the network. Truffles must protect its users from these dangers.

Another security concern is that Truffles must ensure that only data that is explicitly shared be made available to other sites. The non-Truffles portion of each site's file hierarchy must be unavailable to remote sites, and the Truffles portion of the file hierarchy must limit sharing to only those volumes that were meant to be shared.

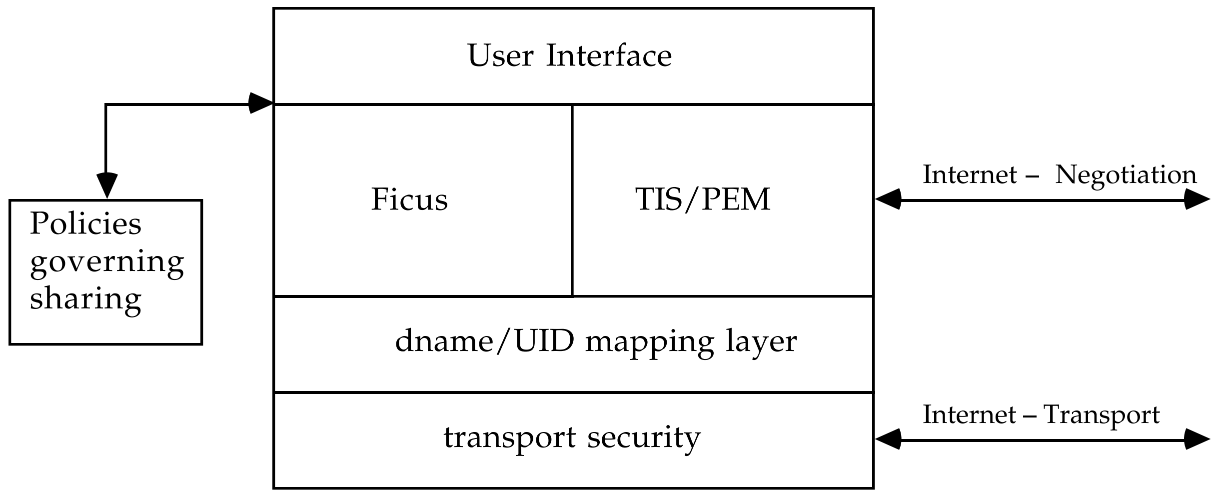

Figure 3. Truffle architecture

Basic Truffles architecture

Figure 3 shows a diagram of the basic Truffles architecture. A user interface hides most of the difficult details of establishing the sharing relationship from the user. This interface is only used during establishment of the relationship; once established, the user works directly through Ficus, which looks like a standard Unix file system interface. The user interface also interacts with the mechanism controlling sharing policy. The boundaries in the diagram show the basic conceptually separate components of Truffles, with adjacencies indicating which components interact directly.

Requests from the user interface are passed either to TIS/PEM or to Ficus. Those requiring off-site transport go through TIS/PEM. TIS/PEM securely transports requests to the TIS/PEM installation on the other site, which then either calls the appropriate Ficus routine or passes the request up through the user interface.

During normal operation, Truffles requests pass through Ficus. Before they go to a different site, they pass through software that maps UIDs from the source administrative domain to those in the destination's domain. The requests themselves go through Truffles transport security.

More complete description of the Truffles architecture requires further understanding of its major components, TIS/PEM and the Ficus file system.

PEM AND TIS/PEM

Privacy enhanced mail (PEM) is an Internet standard that adds certain simple security services (like message authentication, sender authentication, and message confidentiality) to regular electronic mail. (See [3], [4], [5], and [6] for details of PEM's design.)

PEM authentication is done via public key cryptography. A hash of the message to be sent is encrypted with the sender's private key. The receiver can then check the authenticity of the message by decrypting the hash with the sender's known public key and comparing it to the hash produced by the message itself. RSA public key cryptography is used by PEM for authentication.

Encryption of PEM messages uses the Data Encryption Standard (DES) and RSA. An encrypted PEM message consists of the message encrypted with a DES key, the DES key encrypted using the public key of the receiver, and a hash of the message encrypted with the private key of the sender. The receiver uses his private key to decrypt the DES key, uses that to decrypt the message, takes a hash of the message, and compares it to the hash obtained by applying the sender's public key to the encrypted hash. Every message uses a new DES key for encryption. Public and private key pairs are changed only rarely.

Since PEM relies on using public keys to authenticate messages, it must make public keys securely available to its users. Users must be able to dependably ask for and receive a public key for some other user who has sent them mail. X.509 certificates are used for this purpose. These certificates bind a name (an X.500 distinguished name) to a public key. The binding is vouched for by a Certification Authority (CA). Certification Authorities themselves are vouched for by higher level Authorities. PEM makes use of a certification hierarchy in the form of a tree, where each node is certified by a node above it, and the leaves of the tree are users, mailing lists, etc. The Internet Policy Registration Authority (IPRA) is at the highest level of this hierarchy. This authority will be managed by the Internet Society.

TIS/PEM is a reference implementation of the PEM standard, developed by Trusted Information Systems [7]. It is UNIX based, and runs on a variety of platforms. Figure 4 shows a view of TIS/PEM. The PEM library serves as the primary entry point to the system by electronic mail or other services. That library, certain PEM utilities, and key management programs communicate with the local key manager (LKM), which handles key management, independent of the particular application requesting its services. The LKM maintains a local database for certificates and private keys, enforces access control, and provides cryptographic services employing private keys. One of the private libraries attached to TIS/PEM is the crypto library, which has an algorithm independent interface, and handles key generation, message digest computation, encryption and decryption, and signature computation and verification for a variety of encryption schemes.

TIS/PEM's role in Truffles is to provide secure electronic mail services, which will be used to perform the setup of file sharing relationships. Also, TIS/PEM's key manager services are used to handle encryption keys related to the secure transport of data between sites sharing files through Truffles.

TIS/PEM is currently in use at a variety of sites, including three TIS sites spanning the country, UCLA, and others.

THE FICUS FILE SYSTEM

Ficus is a distributed file system designed to run on networks of Unix systems, ranging from portable units and workstations to large file servers [2]. Ficus provides high availability for read and update, utilizing an optimistic "one copy availability" policy. "One copy availability" permits access to a file even if a majority, quorum, or token are unavailable, as long as a single copy can be accessed. This policy maximizes availability, at the cost of permitting copies of a file to become conflicted, when different copies are updated simultaneously while not in communication. Ficus handles such conflicts by reliably detecting them. Many conflicts are automatically resolved by Ficus.

Ficus supports very high availability for both read and write, allowing uncoordinated updates when at least one replica of the file is available. No-lost-update semantics are guaranteed. Asynchronous update propagation is provided to accessible copies on a "best efforts" basis, but is not relied upon for correct operations. Rather, periodic reconciliation ensures that, over time, all replicas converge to a common state. This policy is more appropriate than serializability for the scale and failure modes of a very large distributed system.

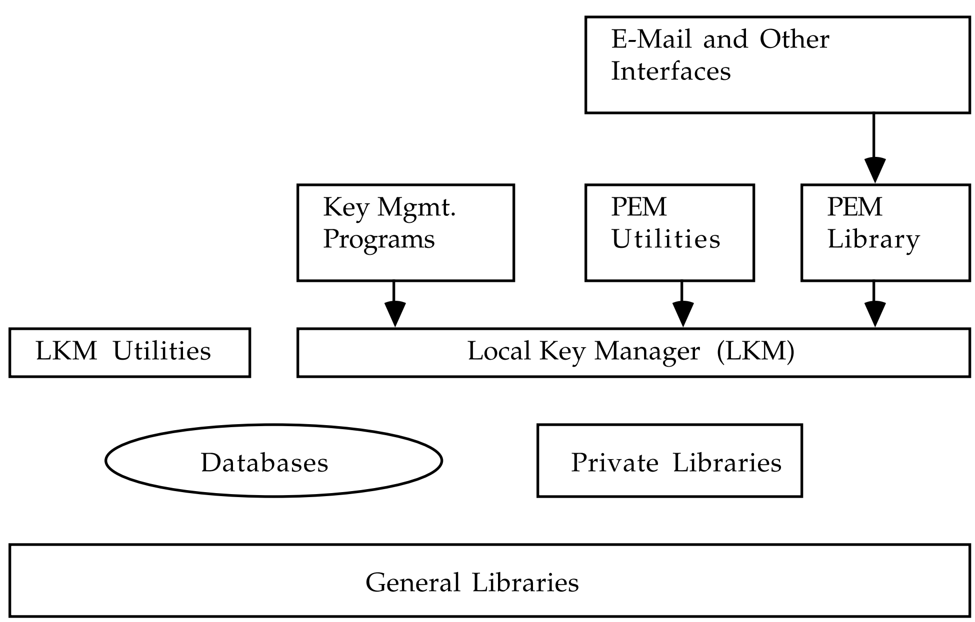

Figure 4. TIS/PEM architecture

Both because of the asynchronous update strategy and the one copy availability policy, different replicas of Ficus files can become in conflict. Conflicts occur when two or more replicas all receive updates without successfully propagating their updates to the other replicas. Conflicts are reliably detected and directory update conflicts automatically reconciled. Many other types of file conflicts are also automatically reconciled. Those conflicts that cannot be resolved automatically are brought to the attention of the owning user for resolution. Ficus provides tools for users to reconcile such conflicts by hand. Experience with Ficus has shown that conflicts are relatively rare events, and are generally easy for users to reconcile.

Ficus is built to run in a single administrative domain. It assumes that all sites and the connecting network are all trusted, so no special security is necessary. Moreover, it is not prepared to deal with sites that have different sets of users with conflicting user identifiers. With some effort, Ficus can work in this environment, but it requires substantial work by the various installations' system administrators, and is less than perfect in many other ways.

Ficus and stackable file systems

The replication service of Ficus is packaged so that it may be inserted above the base Unix filesystem on any machine running a stackable file system interface. This modular architecture permits replication to co-exist with other independently implemented extended filing features.

In addition to running on top of stackable file systems, Ficus is built using stackable layers [8]. The stackable layers approach to file system design permits adding functionality to an existing file system merely by writing the new functionality into a new layer of code. This code is placed on top of the existing layers, providing a compatible interface to users, while simultaneously making the new functionality available. The stackable layers approach does not require any changes to the existing code, so adding functionality is relatively easy.

Ficus itself consists of two layers that sit on top of the Unix file system (UFS) and the network file system (NFS). The Ficus Physical layer supports operations that deal with a single replica of a file. The Ficus Logical Layer supports operations that deal with all replicas of a file. The UFS provides actual storage of data on disk, and NFS is used as a transport layer to move Ficus requests from one site to another. Figure 5 shows a typical stack of Ficus layers on two sites.

A great advantage of the stackable layers technology is that other filing services can be used in conjunction with Ficus, merely by inserting another layer into the appropriate place. Encryption and compression of files are two examples of services that could be combined with Ficus via layers.

Ficus was built at UCLA, and is in daily use there, as the system on which further Ficus development work is done. Ficus has also been installed at several other sites, including TIS and ISI.

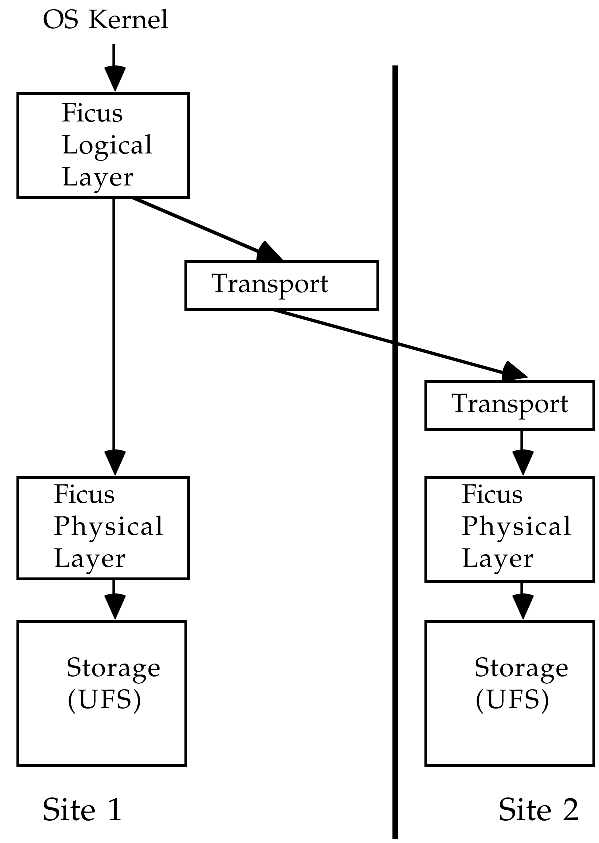

Figure 5. A Ficus stack

HANDLING SECURITY PROBLEMS

Truffles faces several security problems:

- Security of the setup procedures
- Security of the ongoing sharing relationship
- Proper handling of different administrative domains
- Enforcement of policies governing sharing

Secure setup of Truffles sharing relationships

There are several security problems relating to setting up sharing relationships. The users participating in the relationship want to be sure that the others are who they claim to be. Otherwise, an intruder could masquerade as another user to gain improper access to files he otherwise wouldn't be permitted to share. The protocol's various messages must each be authenticated, both to ensure that an intruder is not inserting protocol messages for his own purpose, and to ensure that the participants are not improperly trying to gain access to more than the files that were agreed upon.

When two users decide to share a file volume via Truffles, they must first authenticate their identities. They will do so with electronic mail messages sent under TIS/PEM containing their certificates. Each user will check the identity of the other by authenticating the certificate. The user who already has access to the volume (the originator) will then invoke a program to start up the protocol. This program will accept information about which volume is to be shared with which user on which site, to ensure that not only are protocol messages coming in from the proper sender, but that they refer to the proper volume.

The protocol could take any of several forms. One possible protocol would request a session key from TIS/PEM's key management facilities. This session key will be used to authenticate messages in this volume's setup protocol. The Truffles program will then encrypts this session key using the new user's public key, obtained from TIS/PEM's key management facility and send the result to a daemon at the new user's site.

The daemon at the new user's site will receive the message and start the process of creating the volume replica. After extracting the session key and authenticating the message, it will invoke a Ficus program that creates a volume replica. After this Ficus program runs, the replica is empty, and is not logically connected to the original replica, so further work is necessary. The Truffles daemon will get the replica ID of the new replica and send it to the originating site's Truffles daemon, appropriately authenticated.

The originating site's Truffles daemon will check the authentication, and check to see that the request concerns a volume that was supposed to be shared. It will then run a Truffles program to tell the originating site's replica of the existence of a new replica. Once this program completes, the originating site's daemon will send another message to the new site's Truffles daemon.

After checking authentication on this second message, the new site will run a Truffles reconciliation program that pulls the contents of the volume's files from the original replica to the new replica. This Truffles program will also add a line to a Ficus system file to ensure that this site will periodically run the reconciliation program on the new volume replica, to ensure that the replica remains up to date.

Later, other sites may join the relationship. The process of adding them will be substantially the same as adding the first site. A single existing member of the relationship will exchange messages with the new site using a protocol similar to that described above. The standard Ficus update propagation and reconciliation mechanisms will ensure that all other sites participating in the relationship are quickly informed of the presence of the new site.

The protocol to establish the relationship will probably use electronic mail and TIS/PEM for all its messages. TIS/PEM provides authentication and encryption, making it unnecessary to introduce another secure transport mechanism for these purposes. The ongoing activities of sites sharing files via Truffles will be handled largely by another secure transport mechanism, built into Ficus, since these activities are not suitable for electronic mail. They will continue to use TIS/PEM's key management facilities, however.

Security of the ongoing relationship

Once a Truffles relationship has been set up, the major security concern is that only legitimate participants be permitted to read or write data. The two sites involved in the relationship can use standard Unix and Ficus access control mechanisms to ensure that only the local users who have proper permissions can gain access to this data. But, since the data handled by Truffles might pass over an insecure medium, more is necessary.

Encryption of the data that Truffles sends from site to site is all that is really required. The sites participating in a sharing relationship must encrypt any Ficus requests they send to each other, using a key particular to the relationship. This key will be generated and distributed during the creation of the relationship. A separate key is used for each relationship to ensure that sites participating in one relationship with other sites cannot improperly eavesdrop on conversations about volumes they do not share.

Encryption between the sites in a relationship is done with DES. A unique initialization vector is attached to each message and encrypted, ensuring that two otherwise identical messages do not encrypt to the same value.

Optionally, for situations in which privacy is not a concern, but authentication is, Truffles allows a volume's traffic to be merely authenticated, rather than encrypted, thereby saving the cost of encryption.

Handling different administrative domains

In a Unix system, users are known by several names, two of which are of importance to Truffles. For login purposes, they are identified by a character string name. For purposes of saving file and process ownership information, they are assigned an integer identifier, commonly called the user identifier, or UID. The system maps between the character string login name and the numerical UID whenever necessary, using information stored in the password file or the NIS.

The mapping of login names to UIDs is only unique to a given administrative domain, which is made up of one or more machines closely connected together. Generally, machines that wish to set up Truffles relationships are not in the same administrative domain. Different users might have the same login name and/or UID in the two domains. Since file ownership and access requests are tagged with the UID, and access permission is checked using the UID, there is a security risk in this situation. Unless Truffles can handle this problem, a user on one site might improperly be given access to files on another site simply because another user on that site has the same UID. The situation is intolerable, even when security isn't a concern, as the potential for user confusion is considerable. There is a similar problem when the system maps from a UID to a login name, as it does when a user wants to display the ownership of a file.

This problem has been recognized before in other distributed file services, such as RFS [9]. Their solution was to map UIDs from remote machines to UIDs on local machines. This method worked reasonably well in RFS, since an RFS file was stored on a single machine, and the ownership information for that file could be stored as the local version of the UID for the owning user on that machine. In a replicated file system like Truffles, replicas of the file might be stored at different sites with different UIDs for the owning user. This situation causes a certain complexity in replication control, as the ownership information from one replica must not be propagated to another under normal circumstances, yet must be propagated upon change of ownership. Also, this situation makes it difficult to move the physical storage for a replica from one site to another, since the UIDs associated with the replica's files might be map to different UIDs on the new site.

Truffles will handle this problem by mapping a user's UID to a globally unique identifier. Truffles will save file ownership information using this globally unique identifier, storing it as one of the file's attributes. Truffles will map from the UID to the globally unique identifier whenever a user process tries to access a file. The requesting user's globally unique identifier can then be compared to the file owner's identifier to determine if access should be granted. When the system needs to perform the reverse mapping, to display the ownership of a file, Truffles will map from globally unique identifier to login name. Since the file has a single globally unique owner at all replicas, it will be easy to handle update propagation, and the physical storage can be easily moved from one site to another without losing ownership information.

Many forms of globally unique identifiers could be used for this purpose. Truffles will initially use X.500 *distinguished names* (dnames). When a user makes a request for a file stored under Truffles, his local UID will be mapped to a dname, which is used to determine whether he can access the file. Should the request require remote access at the other site, the dname must be passed with the request across the net. On the opposite end, Truffles will compare the dname to the owner of the file in question, in the same manner as the local case.

This mapping of UIDs to dnames and access permission checking will be done via a Truffles file system layer that sits above the Ficus logical layer. All requests for Truffles volumes will go through this layer, and access permissions will be checked before the request is submitted to the lower layers of the file system. Those lower layers will never reject a request that has been approved by the Truffles layer, since access checking has already been done. Because Ficus has a layered file system available, no existing system or application code will need to be changed to make access checking via dnames work. All the new code will exist in a self contained layer and associated new utility programs.

Certain UIDs in Unix systems have special meanings, especially the root user ID. The root user on a system is permitted to perform many operations that could have disastrous consequences if done improperly. Also, the root user can effectively gain access to any files on the site. Generally, no remote user should ever be permitted to map to the root user on a site through Truffles mechanisms. In particular, the root user on site A should not be able to use a Truffles sharing relationship to gain root privileges on site B. Truffles will not permit any remote dname to be mapped to the local root user. Attempts to do so, even by privileged users, will be rejected by the Truffles software.

An analogous problem exists with group access permissions. Unix systems permit users to belong to several groups, and group membership can also allow access to files. Like UIDs, group IDs (GIDs) are numerical, and are not coordinated between different administrative domains, so a given GID in one domain might refer to an entirely different group in another domain. The Truffles solution to this problem will be similar to its solution for UIDs. Users will be permitted to establish groups in one domain, and users in other domains can map that foreign group to a local one. A common case is expected to be two newly cooperating users setting up a special group strictly to permit them to jointly access their shared files, while using Unix access control to lock others out. Truffles will include tools that make this common case simple to set up.

Truffles has not, as yet, dealt substantially with the issue of revocation of access. In the simplest, probably most common, case, a temporary sharing relationship will come to an end, and must be cleanly torn down. Less frequently, users who are currently able to use files in a sharing relationship must be prevented from doing so in the future. The former case will be adequately dealt with by standard Truffles methods of destroying volumes. The latter case is a subject of further study for Truffles.

Enforcement of policies governing Truffles sharing

The purpose of Truffles is to permit users at different sites to share files with each other without undue burden on system administrators. However, system administrators still need to exercise some control over which of their sites' files can be shared, by whom, with whom. Certain files may be sufficiently sensitive that no one should be permitted to share them outside the site. Others might only be sharable by certain trusted users, with other local users unable to set up sharing relationships on them. Yet others might be sufficiently insensitive that any user who can access them should be permitted to share them.

Generally, the sharing policies that different system administrators may want to enforce could be quite varied, ranging from freely permitting any sharing relationships to requiring explicit permission for all relationships. Truffles must be able to support the whole spectrum of possible policy decisions.

Truffles will support this spectrum of policies by separating the policy from the mechanism. When a user wants to share a volume with another site via Truffles, he will invoke a command to start up the relationship, as described in section 5.1. This command will consult a system file to determine which policy module should be run to determine if the system will permit the relationship to go forward. This system file will be set up by the system administrator, and will indicate what program should be run to determine whether to let the user proceed. Truffles will be distributed with a small set of programs for this purpose, and with instructions on how to write other programs for policies the distribution set does not support. The various programs will require different input information, including user identities, volume identifier, identity of the new site, and possibly certificates of various sorts.

The new user's site must also validate the sharing request, and will do so in a similar manner.

Part of the ongoing Truffles research is to examine what sharing policies are most important, and the best way of implementing them.

RELATED WORK

Truffles is basically a system for sharing files across machine boundaries. The primary related work is other file services with the same goal. One obvious effort is NFS [10]. In fact, the early version of Truffles uses a modified version of NFS as a transport layer. However, NFS has certain limitations that Truffles does not have. Setting up an NFS relationship is a heavyweight operation, requiring substantial system administrator intervention on both sides. Also, NFS currently provides little security (though its security will be improved in the near future). In its original version, NFS did not provide any form of replication service. A subsequent version has provided a form of replication through automounting, but this replication method does not automatically propagate updates [11], making it more suitable for read-only files (like manual pages) than more general file usage. NFS lacks a protocol for automatically setting up the sharing relationship, as well.

Other related systems include the Andrew File System and RFS. The Andrew File System [12] is meant to work in a rather different environment than Truffles. The Andrew File System consists of a distributed collection of servers (known as Vice) servicing a much larger numbers of workstations, each of which runs software known as Virtue. The files are stored permanently by the Vice servers, with extensive caching done by the Virtue clients. The Andrew File System authenticates a workstation and the Vice servers to each other, when they first communicate. Subsequent communications can be encrypted or merely authenticated. Since local copies of the file are cached only, the issue of replication at the client sites does not arise. A given Andrew installation uses a global name space for its users' identifiers, thus avoiding the problem of mapping disjoint identifier spaces. Only workstations that are members of the Andrew File System installation can share files. Thus, the Andrew File System cannot be used to assist arbitrary users at arbitrary sites to share files.

RFS offers a similar service to NFS, with the primary difference being that RFS maintains state for file operations, while NFS does not [9]. RFS has the same general set of limitations as does NFS, for the purpose of solving the Truffles problem. Setting up RFS remote mounts is an administratively heavyweight operation, RFS does not support replication, and RFS does not include a protocol to set up the relationship. RFS has addressed some security concerns that NFS does not, including allowing only specified users to mount file systems, and mapping user and group IDs from other administrative domains.

The Locus Operating System supported replicated files with automatic update and recovery mechanisms [13]. However, Locus ran in a single administrative domain, with all sites in close cooperation. While possible for a relatively small set of machines, this solution cannot apply to the broader case of sharing files with arbitrary users at other sites. Also, since Locus typically ran within a local area network, rather than across long haul lines, and since a single administrative authority controlled the entire system, the security issues that Truffles deals with were not considerations in the Locus system.

Kerberos [14] offers an authentication service that has some overlap with TIS/PEM. Kerberos is specifically designed to authenticate various entities to each other securely. In a Kerberos system, a Kerberos server stores a database of authentication information. Each entity that can be authenticated (referred to as a principal) has a secret key known only to itself and the Kerberos server. Principals authenticate each other through the Kerberos server, which then assigns them a session key to use for encryption during that session. Kerberos names entities using a combination of a primary name, an instance, and a realm; for example, name.instance@realm. Kerberos currently uses its own form of names for principals, rather than X.500 distinguished names. Also, Kerberos itself does not provide for the secure transmission of electronic mail, though its services clearly could be used in a secure mail system. The secure connections provided by Kerberos could also be used to perform setup of Truffles volumes. Kerberos only provides services for authentication. It is not a file sharing or replication facility.

Project Athena makes use of Kerberos as part of its distributed services [15]. Unlike Kerberos itself, Athena is a distributed filing service. Athena uses a workstation/server model for its system, unlike Truffles, in which all sites are viewed as peers. Athena workstations are regarded as dataless nodes, which use their local hard disks to cache data to reduce network traffic. Users log into workstations, are authenticated via Kerberos, and get access to their files through file servers. Athena uses NFS to make remote files available to users. Replication is only supported for read-only system and library software. Athena is intended for use in a single (though possibly very large) administrative environment, like a university. It is not meant to support the more general sharing patterns Truffles supports.

CONCLUSIONS

The resources of the Internet are not well coordinated, largely because of the difficulty of cooperation between its users. The only relatively simple, robust service available is electronic mail. If the Internet, and all other world computer networks, are to reach their full potential, much work is necessary to make cooperation over the networks trivially easy for the average user.

Truffles is intended as a significant step in that direction. It makes one of the primary resources of the network's sites, their file systems, quite easy to share. Users will be able to share files with little more effort than is necessary to send each other electronic mail. The optimistic replication service offered by Truffles' Ficus component will give them high availability, excellent performance, automatic update propagation, and automatic recovery of many conflicts arising from the network environment.

A service like Truffles will never achieve any real popularity, however, unless it is secure. The world network is a dangerous place, and users are rightfully wary about putting too much trust in other sites or the interconnection media. Truffles will use TIS/PEM to reduce the number of entities that must be trusted to a reasonable number — just the certification authorities and the participating sites. By using encryption of data passed via Truffles, privacy of the ongoing relationship will be preserved.

The schedule for building Truffles is relatively short. The plan is dependent almost entirely on reuse of major software components, Ficus and TIS/PEM. While these two system seem entirely unrelated, they can be used together to provide most services required by Truffles. Some time and effort will be spent, of course, in customizing the components for the Truffles environment, and in merging them, but the amount of time saved by not having to re-implement their services is expected far outweigh any time lost merging them. One important result of the Truffles research will be to demonstrate how judicious reuse of existing software components can speed research.

Truffles is currently in an intermediate state of development. Much of the necessary software has been written, and much of the unwritten software has been designed. The major incomplete components are the daemons necessary to run the protocol and the layer of software to perform UID mapping. The basic system should be complete by late in 1993. Further research into the use of Truffles to facilitate sharing between remote users will continue, from that point.

ACKNOWLEDGEMENTS

This work is being performed under DARPA contract number N00174-92-C-0128, under the supervision of Brian Boesch.

REFERENCES

1. Satyanarayanan, M. "A Survey of Distributed File Systems", Annual Review of Computer Science, 1990.

2. Guy, R., Heidemann, J., Mak, W., Page, T., Popek, G., and Rothmeier, D., "Implementation of the Ficus Replicated File System", Proceedings of the Summer USENIX Conference, 1990.

3. Linn, J. "Privacy Enhancement for Internet Electronic Mail: Part I — Message Encryption and Authentication", DEC Technical Report, 1992.

4. Kent, S. "Privacy Enhancement for Internet Electronic Mail: Part II— Certificate-Based Key Management", BBN Communications Technical Report, 1992.

5. Balenson, D. "Privacy Enhancement for Internet Electronic Mail: Part III — Algorithms, Modes, and Identifiers", Trusted Information Systems Technical Report, 1992.

6. Kaliski, B. “Privacy Enhancement for Internet Electronic Mail: Part IV — Key Certification and Related Services”, RSA Data Security, Inc. Technical Report, 1992.

7. Galvin, J. and Balenson, D. “Security Aspects of a UNIX PEM Implemenatation”, Proceedings of the UNIX Security Symposium III, 1992.

8. Page, T., Popek, G., and Guy, R., “Stackable Layers: An Object-Oriented Approach to Distributed File System Architecture”, IEEE Workshop on Object Orientation in Operating Systems, 1990.

9. Rifkin, A., Forbes, M., Hamilton, R., Sabrio, M., Suryakanta, S. and Yueh, K. “RFS Architectural Overview”, Proceedings of the Summer USENIX Conference, 1986.

10. Sandberg, R., Goldberg, D., Kleiman, S., Walsh, D., and Lyon, B., “Design and Implementation of the Sun Network Filesystem”, Usenix Conference Proceedings, Summer 1985.

11. Callaghan, B. and Lyon, T. “The Automounter”, Proceedings of the Winter Usenix Conference, 1989.

12. Satyanarayanan, M., “Integrating Security In a Large Distributed System”, ACM Transactions on Computer Systems, Vol. 7, No. 3, August 1989.

13. Popek, G. and Walker, B., “The LOCUS Distributed System Architecture”, The MIT Press, Cambridge, Massachusetts, 1985.

14. Steiner, J., Neuman, C., and Schiller, J., “Kerberos: An Authentication Service for Open Network Systems”, Usenix Conference Proceedings, Winter 1988.

15. Champine, G., Geer, D., and Ruh, W. “Project Athena as a Distributed Computer System”, IEEE Computer, Sept. 1990.