Truffles – Secure File Sharing With Minimal System Administrator Intervention

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

The Truffles file system supports file sharing between arbitrary users at arbitrary sites connected by a network. Truffles is an interesting example of a service of the future that will automatically allow users to collaborate with other users anywhere in the world in ways not currently possible. These services, and Truffles in particular, have the potential of greatly increasing the workload of system administrators, if the services are not designed properly. This paper describes how Truffles approaches solving its problems without unduly burdening system administrators.

INTRODUCTION

As improvements in hardware and software make new services available, users naturally want to take advantage of them. Frequently, however, a new service means a new burden for system administrators. Beyond the inevitable costs of installing and maintaining the service, however, some services may require more frequent ongoing attention. In particular, a class of services of the future will make resources on remote machines more available to local users, and, conversely, make a given machine's resources more available to remote users. Such services will allow users to customize their own environment to include many remote resources, and to do so easily and quickly. This kind of service has obvious implications for system administrators, as its use means that users will be attempting to share a machine's resources with arbitrary remote users and machines, and will be importing services from equally arbitrary sources. If system administrators are to maintain any control over the security of the computing environment, they must control such sharing. Yet the nature of the service is to set up shared resources at a moment's notice, so explicit system administrator intervention is inappropriate.

A second important implication for system administrators arises from the fact that users will require that their operating system and its associated software provide them with the necessary remote access to services on other machines. In many cases, today's software requires system administrators to take explicit actions whenever such remote access is required. For instance, if a user wishes to mount another site's file system using NFS [1], the system administrator must take a number of non-trivial actions to permit the mount. Clearly, if these services of the future are to achieve their goal of permitting instantaneously customizable user environments, the system administrator must be removed from the loop.

One example of a service with these requirements is the Truffles file system. The Truffles file system permits users at arbitrary sites, with no prior arrangements or connections, to share files with each other. Truffles seeks to solve the problems that limit some existing remote file access tools, such as ftp, telnet, and NFS. These problems include lack of security, high administrator overhead, lack of transparency, and poor performance. Providing a remote file sharing service that avoids these problems requires Truffles to solve several problems, including setting up the file sharing relationship, performance, availability of data, security, and issues of differing administrative domains. A further problem Truffles must solve is ensuring that the system administrators of the machines running Truffles can control what is shared with whom without having to devote too much time and energy to Truffles.

Truffles users will be able to make their files eligible to be shared by calling a single command, and will be able to share others' files by calling another command. Actually setting up the relationship requires much more work than these two commands can perform, however. The command that enables sharing another user's files starts up a protocol between the file owner's machine and new sharer's machine. This protocol runs in the background, and handles all the details of making the files available to the new sharer. The protocol uses electronic mail to transport its messages between the machines, thereby easily gaining advantage of the many solutions electronic mail already provides for working in network environments.

A central problem in file sharing over arbitrary networks is performance. Unless users can access their files in reasonable time, the service is of limited value. Truffles handles this problem through replication. Truffles maintains multiple copies of a shared file so that a user can almost always get the performance of a local file access for a shared file.

Replication also handles the problem of availability for Truffles. Users of shared files want to be able to use those files even when their connections to the other sites sharing the files have been severed. Maintaining a local copy of the file can free read access from dependence on network connections. In many replication strategies, write access is more complex, as the replication system attempts to prevent conflicting updates of a file by only permitting certain copies of the file to be updated when all copies aren't communicating. Truffles uses an optimistic replication strategy, instead. This strategy permits users to update any copy at any time, regardless of whether other copies can currently be contacted. An optimistic replication strategy can lead to update conflicts, but these rarely occur, due to the normal usage patterns of files. Also, Truffles will automatically resolve many conflicts, including all conflicts in directories. Thus, conflicting updates are rarely a problem for users, and can generally be handled easily when they do occur.

Security is a concern both during the process of setting up a file sharing relationship and throughout the life of the relationship. Both the protocol that establishes the shared files between the users and the ongoing network traffic related to keeping the replicas of the files up to date must be protected. Truffles uses a secure form of electronic mail to deal with the former problem, and encrypts the ongoing network traffic to deal with the latter.

Typically, users who share files through Truffles will be in different administrative domains. Each domain keeps its own set of user identifiers, and there is no general mechanism to prevent the identifiers in one domain from overlapping with those in another domain. Therefore, Truffles must be able to recognize user identifiers in their proper context, and must be able to handle user identifiers from a remote domain. Similarly, Unix systems handle much access control through the concept of groups, and the set of group identifiers within one domain may overlap those in another domain. Truffles must deal with them, too.

System administrators need to be able to exercise some control over which files on their machines are shared. Some files may contain sensitive or proprietary data, and should not be made available to users outside the machine. In certain cases, which machine or which remote user is to share the files may be important in a system administrator's decision of whether to permit sharing. Other factors may be important in the decision in other cases. Also, system administrators should not necessarily have to personally approve each and every sharing relationship that the users of a machine try to set up. Truffles must provide a general, powerful mechanism to allow system administrators to express and enforce sharing policy on their machines.

BASIC TRUFFLES DESIGN

Truffles provides file sharing through a secure sharing service usable over normal network connections. Truffles uses a secure electronic mail protocol to set up file sharing. File sharing has normal Unix file semantics once the relationship has been established. Because the connections between sites may have high delay, and the networks or sites may fail, Truffles automatically supports keeping multiple replicas of a file on different machines.

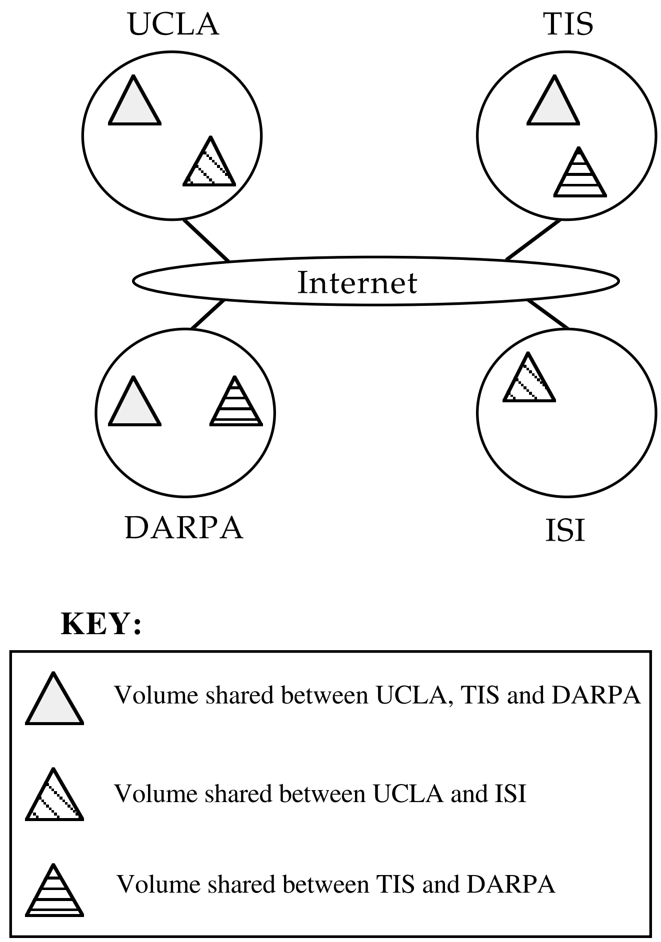

Figure 1. Sites sharing volumes via Truffles

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 a Unix file system. It consists of a connected tree-like structure of directories and files, all stored on a single physical device. Users must collect any files that they want to replicate into a volume or set of volumes. Any files in those volumes are shareable. Normal Unix permission mechanisms control file access. 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 file access permissions and name space organizations, users at the four sites might or might not be able to access files in non-locally replicated volumes.

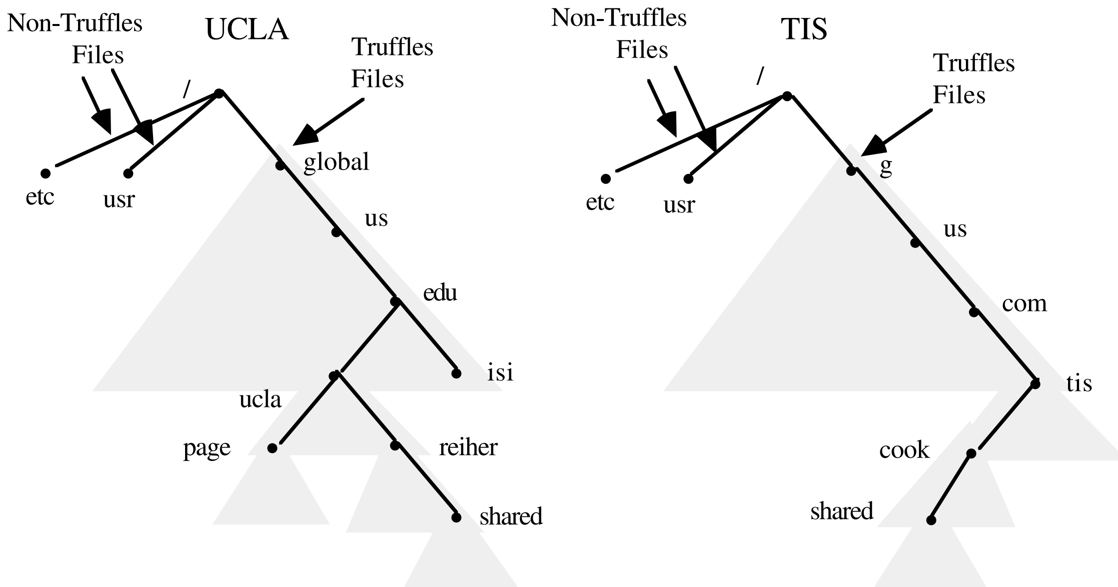

Figure 2. File hierarchies using Truffles

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. Truffles only helps users share files in the Truffles parts of the hierarchies. Files in the non-Truffles parts of the hierarchies are completely inaccessible through 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

`/global`, while the root of the TIS Truffles file system is `/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 can completely share identical Truffles namespaces.

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 access the shared volume remotely, to the extent that normal access permission mechanisms allowed. 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. 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 built from two existing pieces of software. The Ficus file system provides file sharing and replication [2]. The TIS implementation of Privacy Enhanced Mail (TIS/PEM) provides a secure channel for the setup traffic and distributes the keys used for authentication and encryption [3]. Truffles merges these two components, modified in minor ways, with 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 to send the messages that set up relationships, and to authenticate the users. 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 the encryption keys to be used by 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 Truffles servers 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 see the volume at the appropriate places in their local file hierarchies. Ficus ensures that all updates are seen by all replicas. Ficus also deals 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.

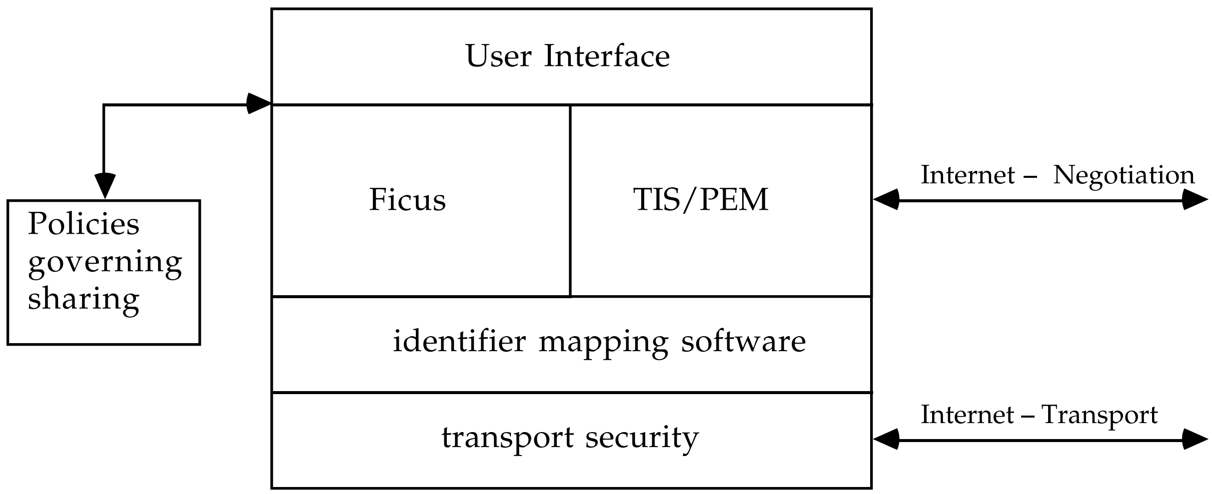

Figure 3. Truffle architecture

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 that controls 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.

The user interface to Truffles consists of three commands – `share`, `participate`, and `withdraw`. `share` is called by a user who wants to make some local files available to users at other sites through Truffles. If the files are not already a volume, they are converted into one, and all preparations are made to share the files through Truffles, including sending notification to any other users who are potential participants. This notification is sent in an electronic mail message to any remote users the local user listed in a file that is one of `share`'s parameters. Once the `share` command completes, the issuing user, and any other users on his machine who normal Unix access control permits, have access to the shared files.

Users wishing to share files that have been prepared for Truffles sharing by the `share` command use the `participate` command. The electronic mail notifications sent by the `share` command contain templates for the `participate` command that must be issued to join in sharing the volume. Users need only fill in one field in the template that specifies where in their local file hierarchy to store the shared volume, and issue the resulting command. `participate` starts up an electronic mail protocol between the Truffles servers on the new user's site and on the site where the `share` command was issued. Upon completion of this protocol, the volume is available for normal Unix use to the users on both sites. Truffles uses electronic mail to inform the participants when the `participate` command completes. Effectively, `participate` sets up a new replica of the shared volume on the calling user's site.

`withdraw` is used to withdraw from a Truffles sharing relationship. Effectively, `withdraw` deletes the local replicated copy of the shared volume on the user's local site. `withdraw` also removes the ability of the site running it to access other replicas remotely.

During normal operation, Truffles requests pass through Ficus. Before they go to a different site, they pass through software that maps user identifiers 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], [6], and [7] for details of PEM's design.)

PEM authentication is done with RSA 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.

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 depend on being able to 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 ([4], [5], [6], [7]), developed by Trusted Information Systems. It is UNIX based, and runs on a variety of platforms. Figure 4 shows an architectural 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.

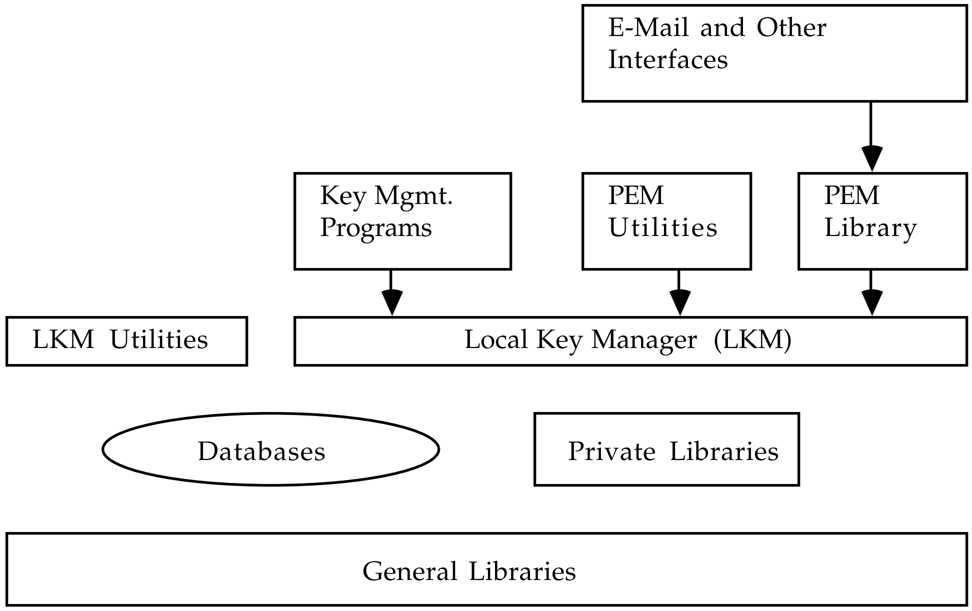

Figure 4. TIS/PEM architecture

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 conflicts among replicas of a file when different replicas are updated while not in communication. Ficus handles such conflicts by reliably detecting them, and, in many cases, automatically resolving them.

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. Ficus provides asynchronous update propagation to accessible copies on a "best efforts" basis, but does not rely upon it 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.

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. Ficus also reconciles many other types of file conflicts. Those conflicts that cannot be resolved automatically are brought to the attention of the owning user or application 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 to reconcile.

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

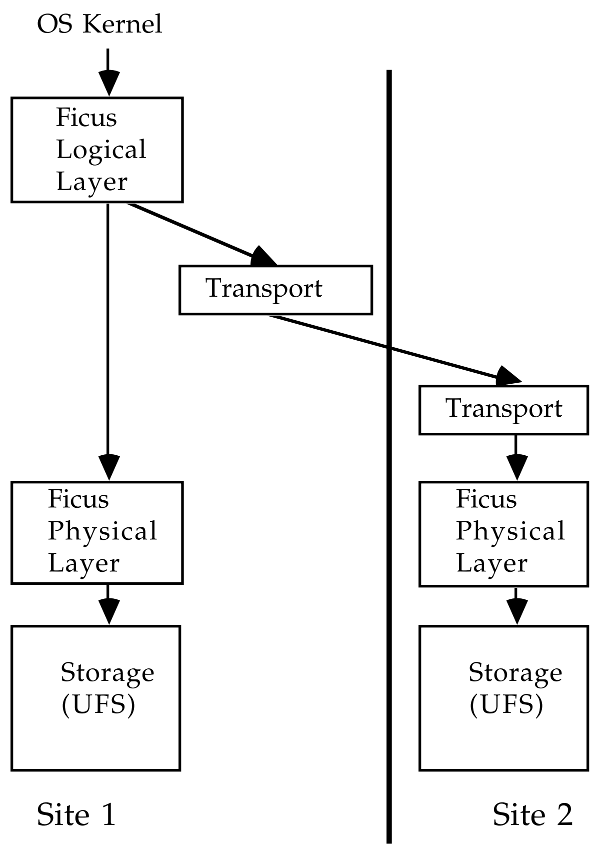

Figure 5. A Ficus stack

SYSTEM ADMINISTRATION PROBLEMS IN TRUFFLES

There are several concerns system administrators are likely to have with the Truffles file system. First, they are likely to have various security concerns, relating both to providing security for Truffles users and ensuring that Truffles does not grant them any privileges they should not get.

A second concern for system administrators on sites running Truffles is maintaining control over what is shared. Since Truffles designers cannot anticipate all forms of control that administrators may need to exercise, Truffles must be prepared to accept a wide variety of different policies.

A third concern is handling user identifiers from different administrative domains. Remote users will have identifiers assigned them by remote domains, not the local domain, and files brought in from remote sites will have non-local owners. The local Truffles system must handle these problems in a way that does not compromise security and allows both users and system administrators to identify who truly owns a file and who should be permitted to access it.

A fourth concern is that Truffles must require minimal ongoing maintenance by system administrators. In particular, Truffles must not burden system administrators with extra work every time a relationship is established.

A fifth concern is auditing. Truffles must maintain an audit trail for all relationships that are set up, so that system administrators can determine what remote access has been granted to their system. Auditing is particularly important since typically Truffles sharing relationships will be set up without the system administrator ever actually seeing a request.

Security in Truffles

Truffles deals with the issues of security of the establishment of sharing and security of the ongoing file sharing separately. Truffles uses encryption and the facilities of TIS/PEM to handle both, however.

Security of the protocol to set up a shared volume between two users at different sites depends on authenticating the parties involved (so interlopers cannot masquerade as legitimate users), authenticating all of the messages used to set up the shared volume (so that interlopers cannot inject false commands into the protocol or play back messages from earlier invocations of the protocol), and guarding the privacy of the protocol (so eavesdroppers cannot find out what is being shared). Both Truffles users and the Truffles servers must have TIS/PEM certificates for authentication. Otherwise, a Truffles server working on the behalf of an interloper could masquerade as a different Truffles server. The TIS/PEM certificate scheme can be used largely as is by Truffles to authenticate both users and Truffles servers.

Truffles uses electronic mail messages for the protocol messages. The Truffles messages can be signed and encrypted by TIS/PEM, thus preventing forgery, replay, and eavesdropping.

The ongoing file sharing relationship requires data to travel between the sites sharing a file. Whenever a change is made to one replica of a file, the update must be propagated to all other replicas to ensure that they have the most up-to-date version of the file. Without some form of protection, interlopers could inject false update messages or alter real update messages, and eavesdroppers could listen to all updates sent over the network. Since Truffles does not use electronic mail to pass updates between nodes, TIS/PEM cannot be used directly to provide security. Instead, the Truffles transport layer of the file system encrypts and signs all outgoing messages. The transport layer on the other end decrypts the incoming messages and checks their validity. Any unencrypted or invalid messages are rejected.

Truffles uses a different key for each volume, so merely sharing a Truffles volume with someone somewhere does not give a site the ability to decrypt all Truffles traffic. Key assignment and management are handled by TIS/PEM's Local Key Manager. Key distribution takes place in the protocol that establishes the shared volume.

Another security concern for Truffles is to limit remote access only to what has been explicitly shared. The Truffles replication method ensures this. Remote users cannot move up the file hierarchy to access files outside the volume they are allowed to access.

Truffles gives no special permissions to remote root users, so a remote root user cannot use Truffles to gain control of a machine. At worst, he can set up Trojan horse programs inside the shared volume. Even those Trojan horse programs, however, will not be given root privileges when they are run, limiting the damage that they can do.

A somewhat more detailed description of how Truffles handles some security issues can be found in [9].

System Administrator Control of Truffles

System administrators need to maintain control of file sharing on the sites they handle. While they may wish to permit many users freedom to share files remotely, some users may not get those privileges, some files may be too sensitive to share off-site, and some remote users and sites may not be trustworthy enough to share with. Therefore, Truffles must provide mechanisms to system administrators to keep file sharing under control. The fact that Truffles file sharing is meant to be set up at a moment's notice at any time makes this problem harder.

The key to the Truffles approach is flexibility. The Truffles designers cannot hope to foresee all possible policies that system administrators may want to enforce, nor can they predict which policies are the most important. However, since Truffles provides a very precise service, there is a limit to the actions that a system administrator could take in response to a request. Basically, the request could be confirmed, allowing the files to be shared; or denied, preventing them from being shared. In cases where files are to be shared with multiple remote users and sites, the system administrator may want to be selective about confirming only some of the requests. In some cases, the system administrator may need to change the decision later, but the most common case is that the decision will be made as the request is considered.

The primary Truffles mechanism for giving system administrators control over file sharing is to allow administrators to insert a policy module of their own devising into Truffles. This policy module takes all available information about the request and returns a yes or no decision. A separate request is made for each remote user/site, so the policy module can be as selective as the system administrator needs it to be.

By using different policy modules, system administrators can consider requests on the basis of the user making them, the files in question, the site or user who the files will be shared with, or any other condition that is important. In some cases, system administrators may defer some or all decisions until they personally inspect the requests. In others, administrators may pre-issue certificates that permit certain types of sharing, and the policy module will examine those certificates. Truffles will provide a few simple policy modules, and tools for administrators to write their own modules, should more complex policies be necessary.

Truffles has not yet dealt with the issue of revocation of sharing. In the simplest case, where all parties agree to end a sharing relationship, the `withdraw` command will work well. In more complex cases, where one or more users lose their access without their consent, `withdraw` is not sufficient. This issue is complex, and is expected to be part of the future Truffles research.

User Identifier Mapping

In a Unix system, users are known by several names. 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 within 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 [10]. 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.

Another approach to solving this problem is mapping a user's UID to a globally unique identifier. Truffles would save file ownership information using this globally unique identifier, storing it as one of the file's attributes. Truffles would 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 could 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 would map from globally unique identifier to login name. Since the file has a single globally unique owner at all replicas, it would be easy to handle update propagation, and the physical storage could be moved easily from one site to another without losing ownership information.

Many forms of globally unique identifiers could be used for this purpose. One option is to use X.500 *distinguished names* (dnames). When a user makes a request for a file stored under Truffles, his local UID would be mapped to a dname, which would be 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 would 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 would be done via a Truffles file system layer that sits above the Ficus logical layer. All requests for Truffles volumes would go through this layer, and access permissions would be checked before the request was submitted to the lower layers of the file system. Those lower layers would never reject a request that has been approved by the Truffles layer, since access checking has already been done. The existing Ficus implementation, however, makes passing of arbitrary data structures (such as dnames) between layers of the file system difficult. Supporting this form of UID mapping would require fairly major modifications to Ficus, as well as new software in the form of a file system layer and some associated utility programs.

Truffles must not allow remote root users to be mapped to the local root user. Attempts to set up such a mapping will be rejected by Truffles.

A problem analogous to the UID 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 probably be similar to its solution for UIDs. Users will be able to set up groups that contain both local and remote users, whatever method is used to map GIDs. 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.

Installation, Updates, and Maintenance

Truffles must be installed by a system administrator, like any other piece of software, and will probably have bug fixes, updates, and improvements. The system administrator will also have to set up some special accounts and obtain TIS/PEM certificates to start Truffles going. However, once Truffles is installed, the system administrator costs are fixed, rather than being proportional to the number of sharing relationships users on the site set up. System administrator need take no actions on a per-relationship basis, so users can set up dozens, hundreds, or even thousands of relationships without further burdening the system administrator.

Despite using NFS as part of Ficus, Truffles makes mounting of remote file volumes easier on system administrators than NFS, because Ficus and Truffles handle all of the work necessary for a remote NFS mount themselves, automatically. The administrator will designate certain ports for Ficus' use, and will never need to concern himself about them further.

Auditing In Truffles

Since Truffles will set up file sharing relationships without bringing them directly to the attention of system administrators, Truffles must maintain complete audit information of all relationships that have been established. Entries will be made in Truffles' logs for each step in each invocation of the protocol that sets up file relationships. System administrators will be able to consult those logs to determine what has been done. In the future, Truffles might also provide tools to allow system administrators to instantly determine which files on sites under their control are being shared with which other remote users. Truffles will not maintain logs of all activity in a shared volume, however.

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 [11]. In fact, 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 [12], making it more suitable for read-only files (like manual pages) than 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 [13] 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. 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 [10]. 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 [14]. 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 [15] 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 [16]. 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.

CURRENT STATUS AND FUTURE PLANS

A test version of the Truffles file system has been set up between TIS' Los Angeles office and UCLA. It is capable of running the `share` and `participate` commands to set up shared volumes between those sites. It lacks any sophisticated error handling, automatic key management, the final mechanism for handling differing user identifier namespaces (a scheme similar to RFS' is used), system administrator control, and certain other features necessary for general use. However, it does set up shared volumes using only these commands, and provides security both for the setup and the ongoing relationship.

The complete version of Truffles will handle more general error conditions, distribute keys automatically, include the final scheme for handling differing identifier namespaces, and provide hooks and a few basic mechanisms for system administrator control of sharing.

The future plans of the Truffles project include examination of system administrator control policies, richer access control mechanisms than those provided by the Unix file system, revocation of file access, and storing encrypted versions of a volume at untrusted backup sites.

CONCLUSIONS

The Truffles file system will offer dramatic new capabilities to users without imposing significant costs on system administrators. Users will be able to share files with other users at arbitrary sites without ever bothering their system administrators or increasing their workload. In this way, Truffles is a model of how to add functionality to a system without adding burdens for the system administrator.

The Truffles research effort is committed to providing Truffles' services securely. Sites running Truffles will not face significant new security risks due to their use of the software, and users of Truffles will have reasonable confidence that their files are shared only with the users and sites they have approved.

Development of Truffles started less than a year ago. Truffles will be built in an unusually short time due to judicious reuse of existing software components. By using the file replication services of the Ficus file system and the secure electronic mail services of TIS/PEM, substantial amounts of Truffles software were essentially already written. The Truffles approach minimizes the changes necessary in these existing components, so the development of Truffles will consist largely of software that works in conjunction with Ficus and TIS/PEM.

Truffles is currently in an intermediate state of development. A version of Truffles with limited functionality is in place for testing purposes. The major remaining component is 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. Galvin, J. and Balenson, D. "Security Aspects of a UNIX PEM Implementation," Proceedings of the UNIX Security Symposium III, 1992.

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

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

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

7. Kaliski, B. "Privacy Enhancement for Internet Electronic Mail: Part IV — Key Certification and Related Services," RSA Data Security, Inc. Technical Report, 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. Reiher, P., Page, T., Popek, G., Cook, J., and Crocker, S., "Truffles – A Secure Service For Widespread File Sharing," Proceedings of the PSRG Workshop, February, 1993.

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

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

12. Callaghan, B. and Lyon, T. "The Automounter," Proceedings of the Winter Usenix Conference, 1989.

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

14. Popek, G. and Walker, B., *The LOCUS Distributed System Architecture*, The MIT Press, Cambridge, Massachusetts, 1985.

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

16. Champine, G., Geer, D., and Ruh, W. "Project Athena as a Distributed Computer System," IEEE Computer, Sept. 1990.