-- --

Access Control in a Workstation-Based Distributed Computing Environment

C.J. Antonelli
cja@citi.umich.edu

W.A. Doster
billdo@citi.umich.edu

P. Honeyman
honey@citi.umich.edu

Center for Information Technology Integration
The University of Michigan
Ann Arbor

Abstract

This paper describes the mechanisms employed to control access to system services on the IFS project. We base our distributed computing environment on systems that we trust, and run those systems in physically secure rooms. From that base, we add services, modifying them to interoperate with existing access control mechanisms. Some weaknesses remain in our environment; we conclude with a description of present vulnerabilities and future plans.

Introduction

The Institutional File System (IFS) project is a joint project of the Center for Information Technology Integration at the University of Michigan and the IBM Corporation. The goal of the IFS project is to offer network file services that are institution-wide, transparent, reliable, and secure. The initial implementation of IFS employs centralized file servers running on mainframes. The project has several dozen staff and a comparable number of workstations.

Like any computing environment, security risks are a major concern, and play a large role in the design of our system. Because they rely almost entirely on network services, workstations at IFS are exposed to a variety of threats. Following Voydock and Kent [1], we classify threats into three categories:

- unauthorized release of information,
- unauthorized modification of information, and
- unauthorized denial of resource use.

At a coarse-grained level, our access control mechanisms attempt to prevent all three categories of potential security violations. At some fine-grained level (say, at the level of our physical networks), the third category poses problems beyond the scope of this paper, and indeed beyond our abilities to address them. However, to the extent that resources, *i.e.*, services, are offered by the project staff, unauthorized denial of resources is addressed by our access control mechanisms.

Bellovin [2] identifies several areas in which networked workstations running the UNIX operating system are vulnerable to attack by outsiders; his analysis is at the level of individual packets. While that is of concern to us, our attention in this paper is to the more coarse-grained problem of how to prevent unauthorized users from invoking processes, obtaining file system access, or denying these to legitimate users.

In general, our goal in securing services is to make it more difficult to compromise the system through network attacks than it is to compromise physical security or to employ social engineering.†

In the remainder of the paper, we start with a logical overview of access control at the IFS project, followed by an enumeration of the services on which the project relies, their interdependencies, and the mechanisms by which access to these services is secured. We conclude with observations on what we do right, what we do wrong, and what remains to be done.

† ‘‘**social engineering:** A nontechnical means of gaining information simply by persuading people to hand it over. If a hacker wished to gain access to a computer system, for example, an act of social engineering might be to contact a system operator and to convince him or her that the hacker is a legitimate user in need of a password; more colloquially, a con job.’’ [3]

Overview

Logically there are two categories of system access that must be protected: invocation of processes and operations on the file system. In controlling who can do what, we face the problems of authentication (who) and authorization (can do). We now discuss how authentication and authorization issues are addressed for these two categories.

Process invocation

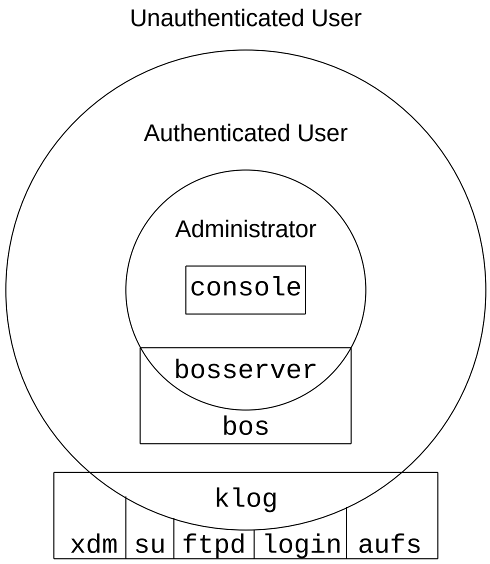
We define three classes of authorization: unauthenticated, authenticated, and administrator.

- Unauthenticated users cannot invoke any processes except those necessary to log in and authenticate.
- Authenticated users can invoke processes on workstations.
- Administrators can invoke processes on system server machines.

A user is initially unauthenticated, and therefore unauthorized. To move to the authorized class, the user must successfully authenticate via one of several programs, *e.g.*, `xdm`, the X display manager; `login`, which is used by `getty` and `telnetd`; `su`; `ftpd`; and some others. All of these programs invoke an application called ‘‘`klog`’’, which authenticates the user with the Kerberos system (explained later in the ‘‘Authentication Service’’ section). If the authentication attempt succeeds, the user is logged in as a normal user.

Administrator authorization is a little involved and is covered further in the ‘‘File Server’’ section. For now, it is sufficient to say that Administrators run `bos`, causing a `bosserver` process on a secure server machine to invoke processes there.

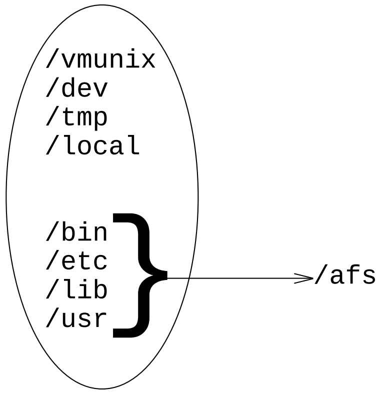
Another way to invoke processes on a server machines is through the console. The following illustration depicts the above arrangement.



File System Access

Control of file system access is a little more complicated. In the IFS environment, we configure workstations to be dataless clients of AFS, the Andrew File System [4,5]. A few files are accessed through the UNIX file system; everything else is accessed from an AFS File Server machine (we say ‘‘in `/afs`’’).

An IFS workstation has a copy of UNIX, `/dev`, `/tmp`, and `/local` on the local disk, amounting to around three megabytes; `/bin`, `/etc`, `/lib`, and `/usr` are symbolic links to directories in `/afs`. Remaining disk space is split between swap and cache for AFS.



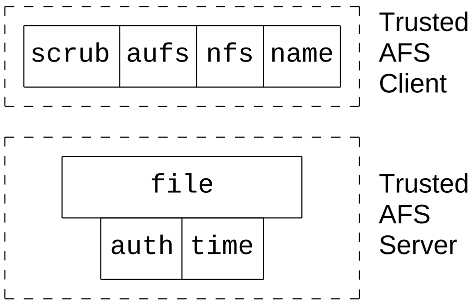
Dataless AFS Client

We describe the mechanism for initializing clients into this configuration in the ‘‘Scrub Service’’ section.

The Servers

Machines providing the services described above run in a locked room. The server machines are divided into two classes: trusted AFS clients and trusted AFS servers. Trusted AFS servers run the absolute minimum necessary to offer AFS file service, which is intended to restrict the space of possible security lapses on these servers. In particular, trusted AFS servers do not run the ordinary complement of network services, such as `inetd`, `telnetd`, `rshd`, *etc*.

On the other hand, a trusted AFS client is configured like any other AFS client machine, so that the lion's share of its files are in `/afs`. All services that require a secure host but are not required to bootstrap an AFS file server are run on trusted AFS clients rather than trusted AFS servers.



This picture shows the organization and serves as a reference point for the sections that follow.

Taxonomy

The previous section presented a logical overview of access control in the IFS development environment. While describing this, several services were mentioned, among them `scrub`, `aufs`, AFS file servers, and authentication. These services in turn depend on time and name service. This section describes each of the above services, giving an overview of its function, authorization model, and dependencies on other services.

Physical security

The basis of trust in our system is physical security. We have a locked room in which we maintain a small number of trusted computing systems. Access to this room is restricted to system staff in possession of a special piece of shaped metal.

We use thin Ethernet, which admits promiscuous snooping, but the strategic direction for campus local networks is twisted pair Ethernet with hubs in locked rooms. The campus backbone is fiber with gateways in secure rooms.

Trusted AFS Servers

From our base of physical security, we deploy the services necessary for supporting the workstation-based computing environment. The most fundamental of these — time, authentication, and file service — are run on trusted AFS servers.

Trusted AFS servers are managed by `bos`, the AFS basic overseer program. `Bos` maintains a list of users permitted to execute privileged commands in the `bos` subsystem. The list of privileged bos users is kept on the server's local disk.

Since one of the privileged `bos` commands can be used to modify the list of privileged users, privilege in `bos` is not to be given nor taken lightly. Another privileged `bos` command allows the issuer to execute a command on the `bos` server. Through a suitable incantation, this command can initiate an interactive shell. It is not strictly necessary to allow anyone to execute a command on the `bos` server in this fashion; it is done for convenience, obviating the need to enter the locked room physically.

Authentication service. We use Kerberos [6], developed by Project Athena [7], for our authentication needs. The Kerberos authentication model is based on a shared secret, which a client uses to prove its identity to a service, and similarly to authenticate a service to a client. These shared secrets are managed by a Kerberos service that listens for requests on a well-known port.

Kerberos provides authentication services to *principals*, which can be users or programs. The Kerberos service keeps a database of its principals; for each principal, it stores a *private key*. This key constitutes the secret shared between Kerberos and the principal.

When a user logs in to a workstation, UNIX initiates a Kerberos authentication session by calling the `klog` program. `Klog` forwards the login name to the Kerberos authentication service running on a trusted AFS server, along with the name of a *ticket granting service* (TGS), also running on a trusted AFS server. The authentication service checks that the user name is valid, creates a *ticket* for the TGS, and encrypts the ticket with a key known only to the TGS and the authentication service. This ticket contains, among other things, a *lifetime*, which limits the length of time that the ticket is valid.

The TGS also generates a random session key, which can be used for secure communication between the client and the TGS. This encrypted ticket and session key are then stored in a response, the response is encrypted with the user's private key, and is sent back to the client. The client then prompts the user for a password, and the password is converted to a private key used to decrypt the response. The encrypted ticket, opaque to the client, and session key are extracted from the decrypted response and stored for future use. The login authentication session is now complete.

When the user wishes to authenticate with a new service, the ticket obtained above is sent to the TGS to obtain another ticket usable for secure communications with the new service. The protocol used for this authentication step is similar to the one outlined above; see [6] for details. `Klog` uses this protocol to obtain a file system authentication ticket.

We don't have a sophisticated white pages service, just a shared `/etc/passwd` file in `/afs`. Because Kerberos provides our authentication service, we store dummy entries in the password field.

Time service. Time service is provided via NTP, the Network Time Protocol [8]. Since Kerberos tickets rely on accurate clocks, reliance on a network time service exposes clients to denial of service attacks. NTP does not offer a facility for remote process invocation, so denial of service is the limit of exposure. NTP has facilities for authentication, although the mechanism for key distribution is rudimentary.

File service. We rely on AFS for distributed file service. Client access to the file system is controlled by Kerberos authentication and access control lists (ACLs). A special kind ACL, the *negative* ACL, removes rights from individuals and groups, which helps administrators react quickly to potential breaches.

The AFS hierarchy is pieced together out of *volumes*. AFS employs quotas to address the resource denial that can arise from malicious or negligent abuse of file system space.

The security and access control characteristics of AFS are detailed by M. Satyanarayanan [9].

Trusted AFS Clients.

Other network services provided to the IFS project are offered on auxiliary hosts under the project's control. Like the file servers, these machines are kept in a locked room, open only to systems staff, and are inaccessible through ordinary network services like `telnet` or `rsh`. The auxiliary servers also run `bos`, so their exposure to unwanted access is like that of the file servers.

Name service. We use `bind` [10] for domain name service (DNS) [11]. DNS is subject to attacks in which the bad guy modifies the mapping between, say, a hostname and an Internet address. We therefore minimize our reliance on hostnames for security, and rely instead on shared secrets. Unfortunately, there are applications and service providers that continue to rely on trustworthy name service, in particular the X window system (see below), and the Berkeley ''`r` commands'': `rsh`, `rcp`, *etc*. Over time, we expect to see these components replaced by Kerberos-based ones. Nevertheless, attacks on DNS can lead to denial of service, so we secure our name servers.

Scrub service. Workstations on the IFS project are dataless, so they rely on AFS for permanent storage. The local disk is used for booting, for storing files specific to the machine, for temporary storage (`/tmp`), and as a disk cache for AFS. Differences between individual workstations, such as the presence or absence of a printer, are reflected in startup scripts stored in AFS.

The `/local` directory on a workstation contains roughly 50 files broken into three categories: booting, workstation-specific, and convenience programs. The boot-related files are those necessary to bootstrap the workstation to the point that `/afs` is made available. The workstation-specific ones are those that need to be different for each machine, *e.g.*, `/usr/adm/messages` is a symbolic link to `/local/usr/adm/messages`. Finally, the `/local/tools/` directory contains files that are useful to have when all file servers are down, *e.g.*, `telnet`, `ftp`, and `vi`. The programs in `/local/tools/` are unused when the file servers are up.

The intent of this design is that any workstation can be ''scrubbed'' clean of its existing local files and loaded with new local files at any time.

Scrubbing entails establishing a network connection with a scrub server and downloading the local files.

Our scrub server runs on a trusted AFS client. The scrub server provides a UNIX `dump` format image of the workstation's local disk to the service requester. This is then piped into the UNIX `restore` command on the requesting machine. We do not enforce any restrictions on who can scrub or which machines can be scrubbed.

File system translation. We offer access to the AFS file system through other distributed file system protocols: NFS [12] and AFP [13]. These ''foreign'' protocols are served by processes running on trusted AFS clients, `nfsd` and `aufs`, respectively.

NFS and AFP have their own authentication models, which we have attempted to integrate into our own. For NFS, we developed an application that allows a user to establish AFS credentials for a third-party NFS server. For AFP, we developed an authentication module integrated with the AppleShare client that similarly establishes AFS credentials for the AFP server.

Normal AFS Clients

Interaction service. Workstations on the IFS project run the X Window System [14]. X has only the most rudimentary sort of access control mechanism. In carefully controlled experiments, we have developed techniques for remotely monitoring a user's activity, including the entering of passwords in windows. These techniques have validated our belief that X should be viewed as a gaping hole in the security structure in the IFS development environment. We are hoping that future releases of X applications take a closer look at security concerns, in particular to Kerberos integration.

Super-user access. Because AFS caches files on the local disk, it is important that local disk access be tightly controlled. Thus it is unwise to allow the super-user password to be widely known, as is done by Project Athena [15, p. 333]. Unfortunately, even without knowing the super-user password, it is usually possible to obtain privileged access to the file system by rebooting the machine into single-user mode. It is therefore important that private files be expunged from the local cache when a user logs out from a machine that is not physically secure.

Discussion

In our academic setting, it is challenging to offer secure yet usable services. Our approach to building trustworthy systems starts with some things that we trust: locked rooms, Kerberos, and AFS; and builds from them other things we trust, name servers, scrub servers, *etc*. However, it is important to be conscious of the vulnerabilities that remain in a system such as ours.

Kerberos appears to be a secure and reliable authentication system. There are, however, some potential problems with it. First, since the Kerberos authentication mechanism depends on the notion of shared secrets, each secret must be stored twice. Kerberos thus suffers from standard key-distribution problems: securely transmitting keys to each site, and ensuring that the two copies of each such key are identical.

Second, Kerberos tickets are written to `/tmp` on the user's workstation. Once gaining root access of the workstation, the bad guy can obtain a copy of any tickets stored in the filesystem and, until they expire, masquerade as the user who obtained them. Kerberos does not destroy its tickets automatically, so a separate administrative step, such as a `.logout` script, is required to remove the tickets. However, if the bad guy is able to gain root access to a workstation while a user is logged in, this step is not effective.

Third, the Kerberos login authentication phase admits dictionary attacks. The bad guy can take a list of likely passwords, convert each to a private key, and use the result to decrypt the authentication server's response to a request for a ticket. From the structure inherent in the server's response, the bad guy can know when the correct password has been used. Furthermore, this dictionary attack can be carried out absent further communication with Kerberos servers. As a prophylactic measure, we employ a Kerberized ''password cracker,'' which doggedly pursues this line of attack.

This last problem with Kerberos authentication makes systems that use it potentially more vulnerable than those relying on traditional UNIX authentication with shadow passwords, and almost as vulnerable as UNIX systems that advertise their encrypted password files, given that the bad guy is able to guess a valid user name.

In our environment, it is easy for a completely unauthorized user to obtain valid user names, simply by inspecting the access control lists of world-readable directories. This inspection can yield both user and group names; for any interesting groups found, the system can be induced to divulge the group membership list. It is also possible to guess interesting group names, *e.g.*, `system:administrators`.

We suggest that the following change to the Kerberos login authentication phase would close this hole: instead of passing back an encrypted response containing some structure, the authentication server returns a nonce identifier. The client is then required to encrypt this nonce with the user's private key, and return to the server. If the server correctly decrypts the nonce, it then provides the usual response.

This method not only foils the bad guy's attempt to decrypt the ticket, because the nonce has no structure and the bad guy can't tell a valid from an invalid decryption, but also involves the authentication server in each decryption attempt. This permits such attacks to be tracked and moderated, possibly by artificially increasing the time required to process each attempt, as is done by the MTS operating system [16].

By virtue of our participation in NAFS, the nationwide file system experiment [17, 18], our file system is accessible to users from universities, laboratories, and commercial interests from Boston to Berkeley. We have not determined the best way to allow users to log in — we now issue unauthenticated access to system binaries. In essence, we are forced to assume that our NAFS colleagues are licensed for the system binaries that we run.

In a similar vein, our home directories are world-readable, so that the Berkeley ''`r` commands'' can read `.rhosts` files kept there. We hope to abandon all reliance on the Berkeley `r` commands, at which time we can restrict access to home directories.

While AFS successfully denies a superuser universal access to its files, obtaining privileged access on workstations or servers permits access to the local disk, bypassing the Kerberos authentication mechanism entirely. It is possible for the bad guy to become superuser on a UNIX workstation by any of several trivial procedures; machines acting as file servers are similarly prone to attack if not locked up. Anyone with superuser access on a file server machine has access to all AFS files stored there. Anyone with superuser access on a client workstation has access to all AFS files cached there. This underlines the importance of physically securing both servers and workstations for total system protection.

Furthermore, it is possible to leave a Trojan horse [19] behind on a workstation, *e.g.*, one that grabs passwords. By minimizing the time required to scrub a workstation, it is conceivable to scrub every public workstation after each use. Scrub time is on the order of five minutes now, clearly too long to be included in the login procedure, but we are investigating scrub optimizations. Of course, this will require a second look at the access control mechanisms employed by the scrub server; a system penetration there would be disastrous.

Conclusion

Our computing environment is not structured from the top down with a security architecture in mind. Instead, it is built piecemeal from services that address our various computing needs. This challenges our ability to build an integrated set of services that works together to protect users and their resources.

As the systems that offer these services mature, they tend to have better access control capabilities. For example, early distributed file system protocols paid scant attention to authentication, but more recent ones offer good protection. Physical security and interaction services are on opposite ends of the spectrum. The former has a history reaching back through the ages, and requires little more effort than drafting a purchase order, while the latter is still quite young and leaves much to be desired in controlling access to resources.

The access control mechanisms employed by distributed services work best when they work together. While this axiom is honored more in the breach than in the observance, we are beginning to see the pieces molding into a unified structure, based on physical security and Kerberos authentication.

Acknowledgements

Gavin Eadie and Bruce Howard built the third-party authenticators for AFP and NFS translation.

We thank Donna Pierson and Ted Hanss for their insights, which improved this paper.

This work was funded by the IBM Corporation.

References

1. V.L. Voydock and S.T. Kent, ‘‘Security Mechanisms on High-Level Network Protocols,’’ *Computing Surveys* **15**(2), pp. 135–171 (June, 1983).
2. S.M. Bellovin, ‘‘Security Problems in the TCP/IP Protocol Suite,’’ *Computer Communication Review* **19**(2), pp. 32–48 (April, 1989).
3. J.P. Barlow, Bluefire, L. Felsenstein, Phiber Optik, Clifford Stoll, and others, ‘‘Is Computer Hacking a Crime?,’’ *Harpers* **280**(1678), pp. 45–57 (March, 1990).
4. M. Satyanarayanan, J.H. Howard, D.A. Nichols, R.N. Sidebotham, A.Z. Spector, and M.J. West, ‘‘The ITC Distributed File System: Principles and Design,’’ pp. 35–50 in *Proceedings of the Tenth ACM Symposium on Operating Systems Principles* (December, 1985).
5. J.H. Howard, ‘‘An Overview of the Andrew File System,’’ pp. 23–26 in *Winter 1988 USENIX Conference Proceedings*, Dallas (February, 1988).
6. J.G. Steiner, C. Neuman, and J.I. Schiller, ‘‘Kerberos: An Authentication Service for Open Network Systems,’’ pp. 191–211 in *Winter 1988 USENIX Conference Proceedings*, Dallas (February, 1988).
7. E. Balkovich, S.R. Lerman, and R.P. Parmelle, ‘‘Computing in Higher Education: The Athena Experience,’’ *Communications of the ACM* **28**(11), pp. 1214–1224 (November, 1985).
8. D.L. Mills, ‘‘Internet time synchronization: The Network Time Protocol,’’ RFC 1129, USC/Information Sciences Institute (October 1989).
9. M. Satyanarayanan, ‘‘Integrating Security in a Large Distributed System,’’ *ACM Transactions on Computer Systems* **7**(3), pp. 247–280 (August, 1989).
10. D.B. Terry, M. Painter, D.W. Riggle, and S. Zhou, ‘‘The Berkeley Internet Name Domain Server,’’ pp. 23–31 in *Summer 1984 USENIX Conference Proceedings*, Salt Lake City (June, 1984).
11. P.V. Mockapetris, ‘‘Domain names — concepts and facilities,’’ RFC 1034, USC/Information Sciences Institute (November, 1987).
12. R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh, and B. Lyon, ‘‘Design and Implementation of the Sun Network Filesystem,’’ pp. 119–130 in *Summer 1985 USENIX Conference Proceedings*, Portland (June, 1985).
13. G.S. Sidhu, R.F. Andrews, and A.B. Oppenheimer, *Inside AppleTalk*, Addison-Wesley, Reading (1989).
14. R.W. Scheifler and J. Gettys, ‘‘The X Window System,’’ *ACM Transactions on Graphics* **5**(2), pp. 79–109 (April, 1987).
15. K. Raeburn, J. Rocjlis, W. Sommerfeld, and S. Zanarotti, ‘‘*Discuss*: An Electronic Conferencing System for a Distributed Computing Environment,’’ pp. 331–342 in *Proceedings of the Winter 1989 USENIX Conference*, San Diego (February, 1989).
16. J.M. Bodwin, ‘‘MTS as a Trusted Computer System,’’ in *Proceedings of MTS Workshop XII*, Rensselaer Polytechnic Institute, Troy (1986).
17. A.Z. Spector and M.L. Kazar, ‘‘Uniting File Systems,’’ *UNIX Review* **7**(3), pp. 60–71 (March, 1989).
18. Information Technology Center, in *Proceedings of the Nationwide File System Workshop*, Pittsburgh (August, 1988).
19. K. Thompson, ‘‘Reflections on Trusting Trust,’’ *Communications of the ACM* **27**(8), pp. 761–763 (August, 1984).