Integrating Security in a Large Distributed System

M. SATYANARAYANAN
Carnegie Mellon University

Andrew is a distributed computing environment that is a synthesis of the personal computing and timesharing paradigms. When mature, it is expected to encompass over 5,000 workstations spanning the Carnegie Mellon University campus. This paper examines the security issues that arise in such an environment and describes the mechanisms that have been developed to address them. These mechanisms include the logical and physical separation of servers and clients, support for secure communication at the remote procedure call level, a distributed authentication service, a file-protection scheme that combines access lists with UNIX mode bits, and the use of encryption as a basic building block. The paper also discusses the assumptions underlying security in Andrew and analyzes the vulnerability of the system. Usage experience reveals that resource control, particularly of workstation CPU cycles, is more important than originally anticipated and that the mechanisms available to address this issue are rudimentary.

Categories and Subject Descriptors: C.2.0 [**Computer-Communication Networks**]: General—*security and protection*; C.2.2 [**Computer-Communication Networks**]: Network Protocols; C.2.4 [**Computer-Communication Networks**]: Distributed Systems; D.2.0 [**Software Engineering**]: General—*protection mechanisms*; D.4.3 [**Operating Systems**]: File Systems Management—*distributed file systems*; D.4.6 [**Operating Systems**]: Security and Protection—*access controls, authentication, cryptographic controls*; E.3 [**Data Encryption**]: *data encryption standard (DES)*; K.6.m [**Management of Computing and Information Systems**]: Miscellaneous—*security*

General Terms: Algorithms, Design, Security

Additional Key Words and Phrases: Access lists, AFS, Andrew, Needham–Schroeder, negative rights, orange book, protection domain, RPC, scalability, trust, UNIX

1. INTRODUCTION

Andrew is a distributed computing environment that has been under development at Carnegie Mellon University since 1983. An early paper [21] describes the origin of the system and presents an overview of its components. Other papers

Andrew is a joint project of Carnegie Mellon University and the IBM Corporation. The author was supported in the writing of this paper by the National Science Foundation (contract CCR-8657907), Defense Advanced Research Projects Agency (order 4976, contract F33615-84-K-1520), and the IBM Corporation (Faculty Development Award). The views and conclusions in this document are those of the author and do not represent the official policies of the funding agencies or Carnegie Mellon University.

Author's address: School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213-3890.

[13, 27] focus on the distributed file system that is the information sharing mechanism of Andrew.

The characteristic of Andrew that has influenced almost every aspect of its design is its scale. The belief that there will eventually be a workstation for each person at CMU suggests that Andrew will grow into a distributed system of 5,000 to 10,000 nodes. A consequence of its large scale is that the laissez-faire attitude towards security typical of closely-knit distributed environments is no longer viable. The relative anonymity of users in a large system requires security to be maintained by enforcement rather than by the goodwill of the user community.

A sizable body of literature exists on algorithms for security in distributed environments. The survey by Voydock and Kent [35] describes many of these algorithms and discusses the basic security problems they address. In contrast, this paper focuses on the design and implementation aspects of building a secure distributed environment. It sets forth the fundamental assumptions on which security in Andrew is based, examines their effect on system structure, describes associated mechanisms, and reports on usage experience.

Although Andrew is no longer an experimental system, it is far enough from maturity that many of its details are still evolving. Rather than trying to describe a moving target, this paper presents a snapshot of Andrew at one point in time. The point of reference is the date of the official inauguration of Andrew, on November 11, 1986. At that point in time, there were over 400 Andrew workstations serving about 1,200 active users. The file system stored 15 gigabytes of data, spread over 15 servers. The system was then mature and robust enough to be in regular use in undergraduate courses at CMU and in demonstrations of Andrew at the EDUCOM conference on educational computing. In the rest of this paper the present tense refers to the state of the system at this reference point. Exceptions to this are explicitly stated.

The paper begins with an overview of the entire system and an identification of its major components. Section 3 then discusses the underlying assumptions and the conditions that must be met for Andrew to be secure. Sections 4 to 7 describe the protection domain, authentication, and enforcement of protection in the distributed file system. Section 8 discusses the problem of resource control. Section 9 underlines the fundamental role of encryption and proposes that encryption hardware be made an integral part of all workstations in distributed environments. Section 10 deals with various other security concerns, while Section 11 examines the ways in which the security of Andrew could be compromised and suggests defenses against some of the possible modes of attack. Section 13 outlines the changes that have occurred since the snapshot presented here. Finally, Section 14 concludes the paper with a recapitulation of its central theme.

2. SYSTEM STRUCTURE

Andrew combines the user interface advantages of personal computing with the data sharing simplicity of timesharing. This synthesis is achieved by close cooperation between two kinds of components, *Vice* and *Virtue*, shown in Figure 1. A Virtue workstation provides the power and capability of a dedicated

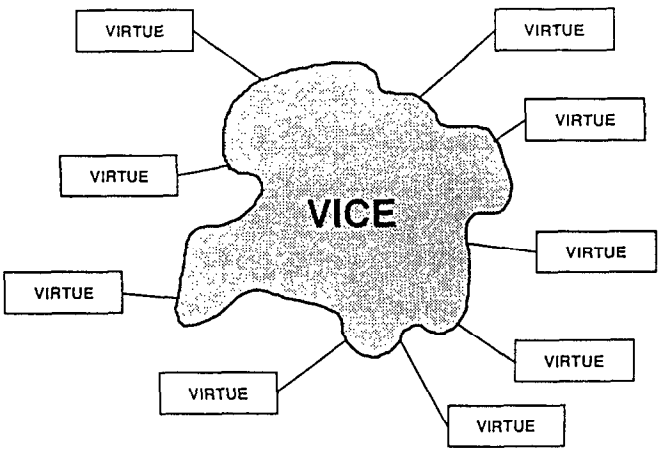

Fig. 1. Vice and Virtue. The amoeba-like structure in the center is a collection of insecure networks and secure servers that constitute Vice. Virtue is typically a workstation, but can also be a mainframe.

personal computer, while Vice provides support for the timesharing abstraction. Although Vice is shown as a single logical entity in Figure 1, it is actually composed of a collection of servers and a complex local area network. This network spans the entire CMU campus and is composed of Ethernet and IBM Token Ring segments interconnected by optic fiber links and active elements called *Routers*.

Each Virtue workstation runs the UNIX 4.3BSD operating system®, and is thus an autonomous timesharing node. Multiple users can concurrently access a workstation via the console keyboard, via the network, or via terminals that are hardwired to the workstation. But the most common use of a workstation, and the usage mode most consistent with the Andrew paradigm, is by a single user at the console.

A distributed file system that spans all workstations is the primary data-sharing mechanism in Andrew. In Virtue, this file system appears as a single large subtree of the local file system. Files critical to the initialization of Virtue are present on the local disk of the workstation and are accessed directly. All other files are in the shared name space and are accessed through an intermediary process called *Venus* that runs on each workstation. Venus finds files on individual servers in Vice, caches them locally, and performs emulation of UNIX file system semantics. Both Vice and Venus are invisible to processes in Virtue. All they see is a UNIX file system, one subtree of which happens to be identical on all workstations. Processes on two different workstations can read and write files in this subtree just as if they were on a single timesharing system.

A mainframe computer that runs Venus appears exactly like a Virtue workstation to Vice. But it is more likely to have multiple concurrent users, who depend on the operating system to protect them against each other. Subverting its operating system is more serious, since more individuals are affected.

® UNIX is a trademark of AT&T Bell Laboratories.

3. ASSUMPTIONS

Saltzer [25] makes an important distinction between a *securable system* and specific *secure instances* of that system. Our purpose in this section is to describe the level of security offered by Andrew and to state the assumptions under which this is achieved. The degree to which a specific Andrew site is secure depends critically on the effort taken to meet these assumptions.

It is easiest to characterize Andrew using the taxonomy introduced by Voydock and Kent. Their survey [35] classifies security violations into unauthorized *release* of information, *modification* of information, and *denial* of resource usage. The security mechanisms in Andrew primarily ensure that information is released and modified only in authorized ways. The difficult issue of resource denial is not fully addressed. The complexity of this problem becomes apparent if one considers a situation where a piece of network hardware is tampered with such that it floods the network with packets. The resulting denial of network bandwidth to legitimate users is clearly a security violation in the strict sense of the term. However, it is not clear what Andrew could possibly do in such situations except to bring the problem to the attention of system administrators. This issue of resource control is discussed at length in Section 8.

Alternative taxonomies of security also exist. Wulf [37], for instance, considers the security of the Hydra operating system in the light of the problems of *mutual suspicion, modification, conservation, confinement,* and *initialization.* It is more difficult to characterize Andrew within this framework. Since Vice and Virtue do not trust each other until a user successfully executes the authentication procedure described in Section 5, there is indeed mutual suspicion. But users do depend on Vice to provide safe, long-term storage of their files and to enforce their protection policies. Andrew can protect against modification of files by other users, but there is no safeguard against incorrect modifications by Vice itself. Since Andrew supports revocation, it does address the problem of conservation. But the problem of confinement, extensively discussed by Lampson [18], is one that Andrew makes no attempt to solve. It is not clear how the initialization problem in Wulf's model applies to Andrew.

The Department of Defense taxonomy of computer systems [10] classifies computer systems into four major categories with numerous subcategories. Security ranges in strength from class D (*minimal protection*) to class A2 (*verified implementation*). In this classification scheme, Andrew meets the criteria for class C1 (*discretionary security protection*). It comes close to meeting the criteria for class C2 (*controlled access protection*), the main deficiency being in the area of audit. Although Andrew does log authentication failures and modifications to the authentication database, it does not maintain audit trails of all events specified in Class C2.

For simplicity, we restrict our attention in the rest of this paper to the model put forth by Voydock and Kent. We do recognize, however, that a complete analysis of Andrew security in terms of a variety of taxonomies would be a valuable exercise in itself.

A fundamental assumption pertains to the question of who enforces security in Andrew. Rather than trusting thousands of workstations, security in Andrew is predicated on the integrity of the much smaller number of Vice servers. These

servers are located in physically secure rooms, are accessible only to trusted operators, and run trusted software. No user software is ever run on servers. For operational reasons, it is necessary to provide utilities that can be run on servers to directly manipulate Andrew file system data. These utilities can be run only by superusers on servers.[1] Both access to servers and the ability to become a superuser on them must be closely guarded privileges.

Workstations may be owned privately or located in public areas. We assume that owners may modify both the hardware and software on their workstations in arbitrary ways. It is therefore the responsibility of users to ensure that they are not being compromised by software on a private workstation. Such a piece of software, referred to as a *Trojan horse* [12], can be trivially installed by a superuser. Consequently, the user has to trust every individual who has the ability to become superuser on the workstation. A user who is seriously concerned about security would ensure the physical integrity of his workstation and would deny all remote access to it via the network.

In the case of a public workstation, it is assumed that there is constant surveillance by administrative personnel to ensure the integrity of hardware and software. It is relatively simple to visually monitor and detect hardware tampering in a public area. But it is much harder to detect a miscreant becoming superuser and installing a Trojan horse. Keeping the superuser password on a workstation a secret is not adequate because workstations can be easily booted up standalone, with the person at the console acquiring superuser privileges. An organization that is serious about security would have to physically modify workstations so that only authorized personnel can boot up public workstations standalone. At the present time, public workstations at CMU do not have such physical safeguards.

It is common for a pool of private workstations to be used by a small collection of users. Workstations located in shared offices or laboratories are examples of such situations. From the point of view of security, such workstations are effectively co-owned by all users who can physically access them. It is their joint responsibility to ensure the integrity of the hardware and software on the workstations.

It should be emphasized that the preceding discussion of software integrity on workstations pertains to local files. There are usually only a few such files, typically system programs for initializing the workstation and for authenticating users to Vice. All other user files are stored in Vice and are subject to the safeguards discussed in Section 6.

The network underlying Andrew has segments in every building at CMU, including student dormitories. It is impossible to guarantee the physical integrity of this network. It can be tapped at any point, and private workstations with modified operating systems can eavesdrop on network traffic. A consequence of these observations is that end-to-end mechanisms based on encryption are the only way to ensure secure communication between Vice and Virtue. These mechanisms are described in Section 5.

[1] The servers also run UNIX 4.3BSD. A "superuser" is a privileged UNIX user free of normal access restrictions.

The routers mentioned earlier are dedicated computers that run specialized software. The integrity of these routers is not critical to Andrew security. Because Andrew uses end-to-end encryption, a compromised router cannot expose or modify information that is transmitted through it. At worst, it can cause packets to be misrouted or modified in ways that cause the receiver to reject them. These are essentially cases of resource denial, which Andrew does not attempt to address completely. Physical damage to a network segment has similar consequences.

Finally, the design of the Andrew file system postulates the use of an independent, secure communication channel connecting all the Vice servers. This is used for administrative functions such as tape backups and distribution of the protection database described in Section 4. This secure channel has to be realized either by a separate, physically secure network or by the use of end-to-end encryption as in the case of Vice-Virtue communication. At the present time, neither of these measures is used at CMU. The "secure" communication channel is the same as the public network, and communication on it is unencrypted.

4. THE PROTECTION DOMAIN

The fundamental protection question is "Can agent X perform operation Y on object Z?" We refer to the set of agents about whom such a question can be asked as the *Protection Domain* [26]. In Andrew, the protection domain is composed of *Users* and *Groups*. A user is an entity, usually a human, that can authenticate itself to Vice, be held responsible for its own actions, and be charged for resource consumption. A group is a set of other groups and users associated with a user called its *Owner*. The name of the owner is a prefix of the name of the group. It is possible to impose meaningful structure in the names of groups, although Andrew ignores such structure. For example, "Bovik:Friends", "Bovik:Friends.CatLovers", and "Bovik:Friends.CatHaters" could mnemonically indicate the purpose of three groups owned by user "Bovik".

Vice internally identifies users and groups by unique 32-bit integer identifiers. An id cannot be reassigned after creation. Such reassignment would require elimination of all existing instances of the id from long-term Vice data structures, an operational nightmare in a large distributed system. User and group names, on the other hand, can easily be changed.

A distinguished user named "System" is omnipotent; Vice applies no protection checks to it. Our original intent was that "System" would play the same role that a superuser plays in the UNIX systems. In practice, we have found it more convenient to define a special group named "System:Administrators". It is membership in this group, rather than authentication as "System", that now endows special privileges. An advantage of this approach is that the actual identity of the user exercising the privileges is available for use in audit trails.[2] We consider this particularly important in view of the scale of Andrew. Another advantage is that revocation of special privileges can be done by modifying group membership rather than by changing a password and communicating it securely to the users who are administrators.

[2] To prevent a system administrator from erasing records of his actions, audit trails have to be maintained on nonerasable media such as write-once optical disks or on hardcopy. We do not do this at present at CMU.

The protection domain includes two other special entities: the group "System: AnyUser", which has all authenticated users of Vice as its implicit members, and the user "Anonymous", corresponding to an unauthenticated Vice user. Neither of these special entities can be made a member of any group. Although the current implementation blurs the distinction between these two entities,[3] we foresee situations where the distinction will become valuable.

Membership in a group can be inherited. The *IsAMemberOf* relation holds between a user or group X and a group G, if and only if X is a member of G. The reflexive, transitive closure of this relation for X defines a subset of the protection domain called its *Current Protection Subdomain* (*CPS*). Informally, the CPS is the set of all groups that X is a member of, either directly or indirectly, including X itself.

The CPS is important because the privileges that users have at any time are the cumulative privileges of all the elements of their CPS. For example, suppose "System: CMU", "System: CMU. Faculty", and "System: CMU. Students" are three groups with the obvious interpretations. If the second and third groups are members of the first, new additions to those groups will automatically acquire privileges granted to "System: CMU". Conversely, it is only necessary to remove a student or faculty member who leaves from those groups in which that person is explicitly named as a member. Inheritance of membership thus conceptually simplifies the maintenance and administration of the protection domain. The scale of Andrew makes this an important advantage.

A common practice in timesharing systems is to create a single entry in the protection domain to stand for a collection of users. Such a collective entry, often referred to as a "group account" or a "project account," may be used for a number of reasons. First, obtaining an individual entry for each human user may involve excessive administrative overheads. Second, the identities of all collaborating users may not be known a priori. Third, the protection mechanisms of the system may make it simpler to specify protection policies in terms of a single pseudouser than for a number of users.

We believe that this practice should be strongly discouraged in an environment like Andrew. Collective entries will exacerbate the already difficult problem of accountability in a large distributed system. The hierarchical organization of the protection domain, in conjunction with the access list mechanism described in Section 6, make the specification of protection policies simple in Andrew. In spite of this, we are disappointed to observe that there are some collective entries at CMU. We conjecture that this is primarily because the addition of a new user is cumbersome at present. In addition, groups can only be created and modified by system administrators. As discussed in Section 13, these problems are being addressed, and we hope that collective entries will soon become unnecessary.

5. AUTHENTICATION AND SECURE COMMUNICATION

Authentication is the indisputable establishment of identities between two mutually suspicious parties in the face of adversaries with malicious intent. In Andrew, the two parties are a user at a Virtue workstation and a Vice server,

[3] Files stored in Vice by an unauthenticated user appear as if they were stored by "System:AnyUser" rather than by "Anonymous."

while the adversaries are eavesdroppers on the network, or modified network hardware or software that alters the data being transmitted.

The authentication mechanism we use is a derivative of Needham and Schroeder's original scheme [22] using private encryption keys. The overall function is decomposed into three major components:

—a *Remote Procedure Call* mechanism that provides support for security,
—a scheme for obtaining and using *Authentication Tokens*,
—an *Authentication Server* that is a repository of password information.

These components, described in detail in the next three sections, are used in the following way. In response to a standard UNIX login prompt at a workstation, the user provides his name and password. The password is used to establish a secure RPC connection to the authentication server. A pair of authentication tokens is obtained from the authentication server and saved by Venus on the workstation. These tokens are used, as needed, by Venus to establish secure RPC connections to file servers. The establishment of a connection is completely transparent to the user, who in particular, does not have to supply the password each time a new connection is made. Virtue seems no different from a standalone workstation to the user.

5.1 Secure RPC

Early in our implementation, it became clear that the remote procedure call package used between Vice and Virtue was a natural level of abstraction at which to provide support for secure communication. Birrell's report on security in the Cedar RPC package [4] independently confirmed the validity of our decision.

The interface of the RPC package is described in detail in the user manual [28]. When a client wishes to communicate with a server, it executes a BIND operation that sets up a logical *Connection*. Connections are relatively cheap to establish and require only about a hundred bytes of storage overhead at each end. A connection can be set up at one of four levels of security:

OpenKimono	neither authenticated nor encrypted;
AuthOnly	authenticated, but packets not encrypted;
HeadersOnly	authenticated and RPC packet headers, but not bodies, encrypted;
Secure	authenticated, and each RPC packet fully encrypted.[4]

Only the last of these four levels provides true end-to-end security in an insecure communication environment. The third level represents a compromise between security and efficiency. Since data is not encrypted, it is vulnerable to release and modification attacks. But since packet headers are encrypted, an intruder cannot interpose requests on a RPC connection. The second level is useful when mutually suspicious entities communicate over a secure channel. The first level provides no security, and is useful only where trusted peers communicate over a secure channel.

[4] Source and destination addresses are not encrypted. Modification of addresses would result in denial of service, a class of attacks we do not address. Exposure of addresses could leak indirect information, but this is part of the confinement problem that we do not address either.

A client can specify the kind of encryption to be used when establishing a connection. The server provides a bit mask indicating the kinds of encryption it can handle, and will reject attempts by a client to use any other kind. This flexibility makes it feasible to equip servers with encryption hardware as well as a suite of software encryption algorithms of differing strength and cost. A workstation owner can make a trade-off between economy, performance, and degree of security in determining the kind of encryption to use. The preferred approach is, of course, to equip all workstations with encryption hardware. Section 9 discusses encryption in greater detail.

For all the authenticated security levels, the BIND operation involves a 3-phase handshake between client and server. The client side of the application provides a variable-length byte sequence called *ClientIdent* and an 8-byte encryption key for the handshake. The server side of the application supplies a procedure, *GetKeys*, to perform key lookup and a procedure, *AuthFail*, to be invoked on authentication failure. GetKeys and AuthFail are invoked by the RPC runtime system on the server side. The actual key lookup mechanism implemented by GetKeys, and the action taken by AuthFail on authentication failure are transparent to the RPC package.

Denoting the encryption and decryption of a string a by key k as $E[a, k]$ and $D[a, k]$, respectively, the steps performed by the RPC package during BIND are described below. Figure 2 describes the same sequence pictorially.

(1) The client chooses a random number X_r and encrypts it with its handshake key, *HKC*. It sends the result, $E[X_r, HKC]$, and ClientIdent (in the clear) to the server.

(2) When the BIND request arrives at the server, the RPC package invokes GetKeys with ClientIdent as a parameter.

(3) GetKeys does a key lookup and returns two keys. One of these keys is a handshake key, *HKS* (which should be identical to *HKC* for successful authentication), and the other is a newly generated session key, *SK*, to be used after the connection is established. If the return code from GetKeys indicates that the key lookup was unsuccessful (as would happen if a user typed in an invalid login name), the BIND request is rejected immediately, and AuthFail is invoked with ClientIdent and the network address of the client as parameters.

(4) Otherwise the server decrypts $E[X_r, HKC]$ with its handshake key, yielding $D[E[X_r, HKC], HKS]$. This should be identical to X_r if the keys match.

(5) The server adds one to the result of its decryption, then encrypts this and a new random number Y_r with its handshake key. It sends the result to the client.

(6) The client uses its handshake key to decrypt this message. If *HKC* and *HKS* match, the first number of the decrypted pair will be $X_r + 1$. If this is the case, the client concludes that the server is genuine. Otherwise the server is a fake and BIND terminates.

(7) The client adds one to the second number of the decrypted pair and encrypts it with its handshake key. It sends the result, $E[(D[E[Y_r, HKS], HKC] + 1), HKC]$, to the server.

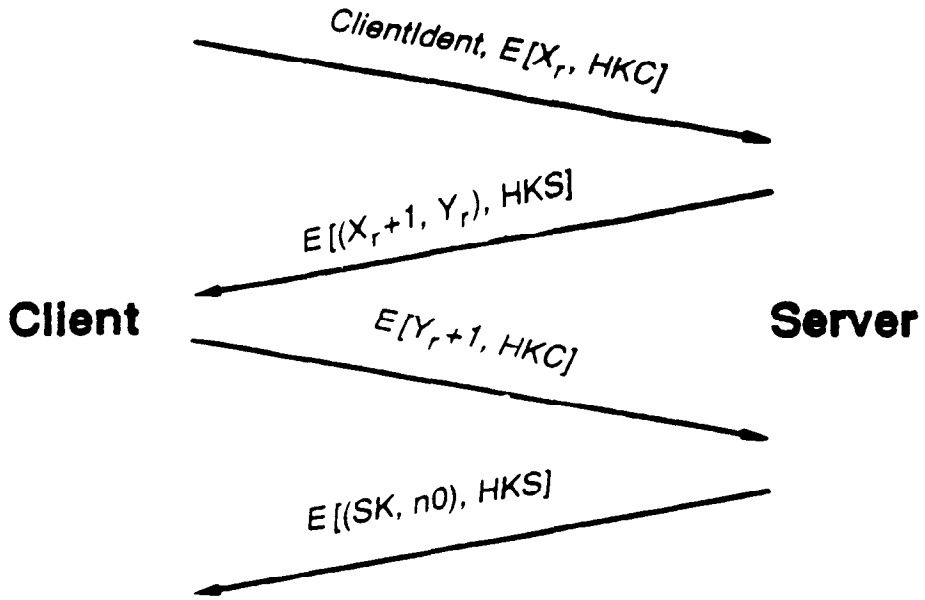

Fig. 2. RPC authentication handshake. This figure shows the sequence of events in the BIND handshake. Each arrow represents a packet. The notation $E[a, k]$ means that a is encrypted with key k. X_r and Y_r are random numbers. *HKC* and *HKS*, the handshake keys of the client and the server, should be identical for a successful BIND. *SK* is a randomly chosen session key, and $n0$ is a randomly chosen initial sequence number. All random entities are chosen afresh for each BIND.

(8) The server decrypts this message with its handshake key. If *HKC* and *HKS* match, the decrypted number will be $Y_r + 1$. In that case the server concludes that the client is genuine. Otherwise the client is a fake and the BIND terminates after AuthFail is invoked.

(9) The server then encrypts the session key *SK* and a randomly chosen initial RPC sequence number $n0$ with its handshake key. It completes BIND by sending the result, $E[(SK, n0), HKS]$, to the client. All future encryption on this connection uses *SK*. The sequence numbers of RPC requests and replies will increase monotonically from $n0$.[5]

Thus, at the end of successful BIND, the server is assured that the client possesses the correct handshake key for ClientIdent. The client, in turn, is assured that the server is capable of deducing the handshake key from ClientIdent. The possession of the handshake key is assumed to be prima facie evidence of authenticity.

The correctness of this authentication procedure hinges on the fact that possession of the handshake key by both parties is essential for all steps of the handshake to succeed. Without the correct key, it is extremely unlikely that an adversary will be able to generate outgoing messages that correspond to appropriate transformations of the incoming messages. Mutual authentication is

[5] Burrows, Abadi, and Needham [5] point out that this step of the protocol should also include the original random number $x0$ encrypted by HKS. This guards against replay attacks by an individual who has broken a previous session key.

achieved because both the client and the server are required to demonstrate that they possess the handshake key. The use of new random numbers for each BIND prevents an adversary from eavesdropping on a successful BIND and replaying packets from that sequence. The presence of an initial sequence number chosen afresh for each BIND defeats replays of the last packet.

Figure 2 summarizes the steps involved in the BIND authentication procedure. It is important to note that the RPC package makes no assumptions about the format of ClientIdent or the manner in which GetKeys derives the handshake key from ClientIdent. The next section describes how this generality is used in Andrew in two different ways: to communicate with an authentication server at login and with a file server when Venus contacts it for the first time. A connection is terminated by an UNBIND call, which destroys every state associated with that connection.

Security in Andrew is not critically dependent on the details of the authentication handshake. The code pertaining to it is small and self-contained. The handshake can therefore be treated as a black box and an alternative mutual authentication technique substituted with relative ease.

5.2 Authentication Tokens

Andrew uses a two-step authentication scheme that is built on top of the RPC authentication mechanism described in the previous section. In the first step, an authentication server is contacted and a pair of authentication *Tokens* is obtained and saved for future use by Venus. In the second step, which occurs each time Venus contacts a new file server, these tokens are used to establish a secure RPC connection for the user. The rest of this section explains what tokens are and describes the details of the two authentication steps.

An authentication token is an object whose possession is proof of authenticity. It is like a *Capability* [17] in that no consultation with an external agency is required when using it, but is different from a capability in that it establishes identity rather than granting rights. Tokens are conceptually similar to *Authenticators* described by Birrell [4].

Tokens come in pairs. One of the components of the pair, the *Secret Token*, is encrypted at creation and can be sent in the clear. The other component, the *Clear Token*, has fields that are sensitive and should be sent only on secure connections. Both tokens contain essentially the same information: the Vice id of the user, a handshake key, a unique handle for identifying the token, a timestamp that indicates when the token becomes valid, and another timestamp that indicates when it expires. The secret token contains, in addition, a fixed string for self-identification. The appearance of this string when decrypting a secret token confirms that the right key has been used. The secret token also contains noise fields that are filled with new random values each time a token is created. This is done to thwart attempts to break the key used for encrypting tokens.

The UNIX program for logging in on workstations has been extensively modified, although its user interface is unaltered. LOGIN now contacts an authentication server using the RPC mechanism described in Section 5.1. The name and password typed in by the user are used as the ClientIdent and handshake key respectively. The GetKeys routine in the authentication server obtains this

password from an internal table. When the RPC handshake completes, a secure, authenticated connection has been established between LOGIN and the authentication server. LOGIN uses this connection to obtain a pair of tokens for the user. The authentication server generates a new handshake key for each pair of tokens it creates. It encrypts the secret token with a key known only to itself and the Vice file servers. LOGIN now passes the clear and secret tokens to Venus, which retains them in an internal data structure. At this point LOGIN terminates, and the user can use the workstation.

Whenever Venus needs to establish a RPC connection to a Vice file server on behalf of a user, it invokes BIND using the secret token for that user as ClientIdent and the key in the clear token as the handshake key. In the first phase of the BIND, the GetKeys routine on the server is invoked with ClientIdent as the input parameter. The server obtains the handshake key from the secret token by decrypting it. The authentication procedure is critically dependent on the assumption that only legitimate servers possess the key to decrypt secret tokens. At this point Venus and the server each have a key that they believe to be the correct handshake key. The remaining steps of the BIND proceed as described in Section 5.1, leading to mutual authentication. If the BIND is successful, the server uses the id in the secret token as the identity of the client on this RPC connection and sets up an appropriate internal state.

Since tokens have a finite lifetime, a user will need to be periodically reauthenticated. At present, tokens are valid for 24 hours at CMU. The program LOG, which is functionally identical to LOGIN, can be used for reauthentication without first logging out. This allows users to retain logged-in context. Users with long-running programs, such as simulations, have to remember to reauthenticate at least once a day. In practice, the 24-hour limit has not been a serious source of inconvenience to our users.

When multiple users are logged into a workstation, Venus maintains a separate secure RPC connection for each of them for each of the Vice file servers they have accessed. When a user logs out of a workstation, Venus deletes his tokens. In the future, Vice may support other services besides a distributed file system. The components of such services which execute in Virtue will be able to use tokens for authentication, just as Venus does at present.

The two-step approach used in Andrew is more convenient and more robust than a single-step authentication scheme for the following reasons:

(1) it allows Venus to establish secure connections as it needs them, without users having to supply their password each time;
(2) it allows system programs other than Venus to perform Vice authentication without user intervention;
(3) it avoids having to store passwords in the clear on workstations; and
(4) it limits the time duration during which lost tokens can cause damage.

5.3 Authentication Server

The authentication server, which runs on a trusted Vice machine, is responsible for restricting Vice access and for determining whether an authentication attempt by a user is valid. To perform these functions it maintains a database of password information about users. An excerpt of this database is shown in Figure 3. The

```
277     545c5058595a5156        aad     Anthony Datri
265     575c585f5b5b575a        ab0q    Alfred Blumstein
672     13020a030619091f        ab2g    A. Leonard Brown
969     5f55595c595e555e        ab6q    Ahmadou Barry
131     565956595f5a545e        abrahams        Julia Abrahams
913     565857585d5a5459        ac2d    Arjun Bijoy Chatterjee
.....
.....
.....
283     13020a030619091f        zubrow  David Zubrow
18      0503135c5a6e676f        # By 18 at Wed Mar 19 13:09:23 1986
18      0503135c5a6e676f        # By 18 at Wed Mar 19 16:36:55 1986
1022    0b0317040709676f        rk27    # By 18 at Wed Mar 19 16:37:37 1986
1023    1500081d190b156f        bd0p    # By 18 at Wed Mar 19 16:37:37 1986
1024    150018030c1d146f        cc37    # By 18 at Wed Mar 19 16:37:38 1986
1025    0b0315021b1d676f        cc38    # By 18 at Wed Mar 19 16:37:38 1986
1026    150f13020502146f        jc15    # By 18 at Wed Mar 19 16:37:38 1986
.....
.....
```

Fig. 3. Excerpt from authentication database. Each entry corresponds to information about one user. The first field is the Vice id of the user, the second is the user's encrypted password; the third field is the name of the user. Other fields are ignored by the authentication server. The first few lines correspond to entries that were present when the database was initialized. The entries at the bottom represent modifications. Each modification is tagged with the identity of the user making the change and the time the change was made.

passwords stored in the database are effectively in the clear, but are encrypted with a key known to the server so that nonmalicious system personnel are prevented from accidentally reading the passwords. This database is used for password lookup whenever a user logs in to a Virtue workstation. It is updated whenever users created, deleted, or have their names or passwords changed. Users can change their own password; other operations can only be performed by system administrators.

Note that it would not be adequate to store a one-way transformation of the password in a publicly readable authentication database, as is done in timesharing systems such as UNIX. That approach assumes that terminals are connected to a mainframe by physically secure lines. The password typed in by a user is securely conveyed to the mainframe, where it is transformed and compared with the string stored in the authentication database. Since the client and the server do not communicate over a secure channel in Andrew, the password cannot be sent in the clear. Further, the UNIX approach does not provide mutual authentication. Although the timesharing system is assured of the user's identity, the inverse is not true. The requirement that passwords (or keys derived from them) be stored in the clear on a server can also be explained by observing that the Needham–Schroeder authentication scheme is built around a shared secret. A publicly readable transformation of the password would not constitute a secret.

Server performance is considerably improved by exploiting the fact that queries are far more frequent than updates. This makes it appropriate for the server to maintain a write-through cache copy of the entire database in its virtual memory. A modification to the database immediately overwrites cached information. The copy on disk is not, however, overwritten. Rather, an audit trail of changes is maintained in the database by appending a timestamped entry indicating the

```
Date:   Mon Sep 29 09:51:13 1986

09:51:13        Server successfully started
11:03:49        Authentication failed for "fs0t" from 128.2.14.11
11:05:22        Authentication failed for "fs0t" from 128.2.14.11
11:05:54        Authentication failed for "an09" from 128.2.14.8
11:09:50        Authentication failed for "wh0s" from 128.2.14.4
11:10:25        Authentication failed for "wh0s" from 128.2.14.4
11:12:28        Authentication failed for "ao07" from 128.2.14.14
11:12:58        Authentication failed for "wh0s" from 128.2.14.4
11:20:43        Authentication failed for "ao07" from 128.2.14.14
12:00:26        Authentication failed for "ks2n" from 128.2.13.3
13:58:46        Authentication failed for "dans" from 128.2.243.3
15:22:26        Authentication failed for "dtla" from 128.2.17.17
16:16:17        AuthChangePassd() attempt on dh2u by js8c denied
16:19:17        AuthChangePassd() attempt on dh2u by js8c denied
16:24:57        Authentication failed for "akll" from 128.2.14.14
16:56:53        Authentication failed for "js8c" from 128.2.17.4
.....
.....
```

Fig. 4. Excerpt from authentication log. This figure shows typical entries from the authentication log. Most of the entries are invalid authentication attempts, probably caused by users typing in their passwords incorrectly. Each entry identifies the user and the workstation from which the operation was attempted. Two of the entries are failed attempts by one user to change the password of another user.

change and the identity of the user making the modification. On startup the authentication server initializes its cache by reading the database sequentially. Later changes thus override earlier ones. An offline program has to be run periodically to compact the database.

The key used by the authentication server for encrypting secret tokens has to be known to all the Vice file servers. This key should be changed periodically if an Andrew site is serious about security. The Vice file servers remember the two most recent such keys and try them one after the other when decrypting a secret token. This allows unexpired tokens to be used even if the authentication server has changed keys. At present, key distribution is manual; this should be automated in the future.

For robustness, there is an instance of the authentication server and database on each Vice machine. All but one are slaves and respond only to queries. Only one server, the master, accepts updates. Changes are propagated to slaves over the secure communication channel referred to in Section 3. For this specific application, nonuniform propagation speed and the temporary inconsistencies that may result do not pose a serious problem. For further robustness, each instance of the authentication server has an associated watchdog UNIX process that restarts it in the event of a crash.

Each server instance has a log file in which authentication failures and unsuccessful attempts to update the password database are recorded. Figure 4 shows an excerpt from such a log. It would not be difficult to provide a more sophisticated and timely warning mechanism for system personnel if suspicious events are observed by authentication servers.

6. PROTECTION IN VICE

As the custodian of shared information in Andrew, Vice enforces the protection policies specified by users. The scale, character, and periodic change in the composition of the user community in a university necessitates a protection mechanism that is simple to use yet allows complex policies to be expressed. A further consequence of these factors is that revocation of access privileges is an important and common operation. In the light of these considerations, we opted to use an *Access List* mechanism in Andrew. The next three sections describe how access lists are implemented, how they are used for file protection, and how Vice represents and maintains information on the protection domain.

6.1 Access Lists

The access list mechanism is implemented as a package available to any service in Vice, though only the distributed file system currently uses it. An entry in an access list maps a member of the protection domain into a set of *Rights*, which are merely bit positions in a 32-bit mask. The interpretation of rights is specific to each Vice service. The total rights possessed by a user on an object is the union of all the rights possessed by the members of the user's CPS. In other words, the user possesses the maximal rights collectively possessed and all the groups of which he or she is a direct or indirect member.

An access list is actually composed of two sublists: a list of *Positive Rights* and a list of *Negative Rights*. An entry in a positive rights list indicates *possession* of a set of rights. In a negative rights list, it indicates *denial* of those rights. In case of conflict, denial overrides possession.

Negative rights are primarily a means of rapidly and selectively revoking access to sensitive objects. Revocation is usually done by removing an individual from an access list. But that individual may be a direct or indirect member of one or more groups that bestow rights on the object. The protection domain has therefore to be modified to exclude the individual from those groups. The process of discovering all groups that the user should be removed from, performing the removal at the site of the master authentication server, and propagating it to all slaves may take a significant amount of time in a large distributed system. Negative rights can reduce the window of vulnerability, since changes to access lists are effective immediately.

As an example, if it is discovered that a member of a group is misusing privileges, that individual can be immediately given negative rights on critical objects used by the group. That member can then be deleted from the group. After the change in membership is effective at all Vice servers, the negative rights entries can be removed. Negative rights thus decouple the problems of rapid revocation, management of the protection domain, and propagation of information in a large distributed system.

Negative rights can be used to specify protection policies of the form "Grant rights R to all members of group G, except user U." The security mechanisms of Multics [25] also allowed the expression of such protection policies. Rabin and Tygar, in their recent work on ITOSS [24], confirm the advantages of providing negative privileges.

The algorithm executed during an access list check is quite efficient. Suppose A is an arbitrary access list and C is the CPS of U. The entries in A and C are maintained in sorted order. The rights possessed by U are determined as follows:

(1) let M and N be rights masks, initially empty;
(2) for each element of C, if there is an entry in the positive rights list of A, inclusive-OR M with the rights portion of the entry;
(3) for each element of C, if there is an entry in the negative rights list of A, inclusive-OR N with the rights portion of the entry;
(4) bitwise subtract N from M;
(5) M now specifies the rights that U possesses.

Profiling of the Vice servers in actual use confirms that the overheads due to access list checks are negligible.

6.2 File Protection

Vice associates an access list with each directory. The access list applies to all files in the directory, thus giving them uniform protection status. The primary reason for this design decision is conceptual simplicity. Users have, at all times, a rough mental picture of the protection state of the files they access. In a large system, the reduction in conceptual state obtained by associating protection with directories rather than files is considerable. A secondary benefit is the reduced storage overhead on servers. Usage experience in Andrew has proved that this is an excellent compromise between providing protection at fine granularity and retaining conceptual simplicity. In the rare instances where a file needs to have a different protection status from other files in its directory, we place that file in a separate directory with appropriate protection and put a symbolic link to it in the original directory.

Seven kinds of rights are associated with a directory:

read(**r**)	read any file,
write(**w**)	write any file,
lookup(**l**)	lookup status of any file,
insert(**i**)	insert a new file in this directory (only if it does not already exist). This is particularly useful in implementing mailboxes.
delete(**d**)	delete any existing file,
administer(**a**)	modify the access list of this directory.
lock(**k**)	lock any file. This has turned out not to be a particularly useful right, but continues to be supported for historical reasons.

The three most commonly used combinations of rights are **rl**, for read access, **rwlidk** for write access, and **rwlidka** for complete access. Figure 5 shows an example of the access list on a Vice directory. Modifications to access lists take effect immediately.

Certain privileges commonly found in timesharing systems do not make sense in the context of Andrew. Execute only privilege, for example, is not a right that Vice can enforce since program execution is done by Virtue. Revocation of read

```
mozart> fs la /cmu/itc/satya/s11
Normal rights:
  System:ITC.FileSystemGroup rlidwk
  System:AnyUser rl
  satya rlidwka
Negative rights:
  System:ITC.UserInterfaceGroup rlidwka
mozart>
```

Fig. 5. Access list on a Vice directory. This figure shows how an access list is displayed in Andrew. The string "mozart)" is the prompt by the workstation. The command "fs la" lists the specified directory. Note the use of negative rights; a member of System:ITC.UserInterfaceGroup would have no rights on this directory, even though System:AnyUser has read and lookup rights.

rights is another area where Vice can do little since Virtue caches files. At best it can ensure that new versions of a file are not readable by the user whose access is revoked.

6.3 Protection Domain Representation

Protection domain information is maintained in a database that is replicated at each Vice file server. The database consists of a data file on disk and an index file that is cached in its entirety in virtual memory. The index file enables id-to-name translations in constant time, and name-to-id translations in logarithmic time. For each entry, the index also contains the offset in the data file where the first byte of information about the corresponding user or group is stored. A typical lookup of the database by user or group name involves a search to find the id, followed by a seek operation and a read operation on the data file.

Each entry in the database corresponds to a single user or group. It consists of a name and an id followed by three lists specifying membership information. The first list specifies the groups to which that user or group directly belongs, while the second list is the precomputed CPS. For a user, the third list enumerates the groups owned by the user; for a group, it is the list of users or groups who are its direct members. Each entry also has an associated access list, which is unused at the present time. We intend to allow users to directly manipulate the database via a protection server. The access lists will then control the examination and modification of group membership. Figure 6 shows an excerpt from the database.

When Venus makes a secure RPC connection on behalf of a user, the file server caches the CPS of the user in virtual memory and uses it on access list checks. At present, changes to the protection domain do not affect the cached copy until the RPC connection is terminated. It would be relatively simple to modify the server to invalidate cached CPS copies whenever the protection database changes.

At present, changes to the protection database are manually performed at a central site in Vice. Utilities are available to simplify the creation or deletion of a user or to modify the membership of a group. These utilities also precompute the CPS by transitive closure and construct the index file. Modifications performed at the central site are asynchronously propagated to all other Vice sites via the secure communication channel mentioned in Section 3. In our experience, the minor temporary inconsistencies that occasionally arise due to varying propagation speeds have not significantly affected the usability of the system.

```
############################
# VICE protection database #
############################

# Lines such as these are comments. Comments and whitespace are ignored.

# This file consists of user entries and group entries in no particular order.
# An empty entry indicates the end.

# A user entry has the form:
# UserName      UserId
#               "Is a group I directly belong to"_List
#               "Is a group in my CPS"_List
#               "Is a group owned by me"_List
#               Access List
#               ;

# A group entry has the form:
# GroupName     GroupId OwnerId
#               "Is a group I directly belong to"_List
#               "Is a group in my CPS"_List
#               "Is a user or group who is a direct member of me"_List
#               Access List
#               ;


# A simple list has the form ( i1 i2 i3 ..... )

# An access list has two tuple lists:
#               one for positive and the other for negative rights:
#               (+ (i1 r1) (i2 r2) ...)
#               (- (i1 r1) (i2 r2) ...)

................................................................................
................................................................................

#               M. Satyanarayanan
satya           19
                ( -201 -207 -209 )
                ( -201 -207 -209 )
                ( -203 -205 )
                (+ (19 -1) (-101 1))
                (- )
                ;

................................................................................
................................................................................
System:UserSupport      -213 777
                ( )
                ( )
                ( 427 177 117 746 585 416 64 201 1032 1247 1244 3017 377 259 172 )
                (+ (777 -1) (-101 1))
                (- )
                ;

................................................................................
................................................................................
```

Fig. 6. Excerpt from Vice protection domain database.

7. PROTECTION IN VIRTUE

As a multiuser UNIX system, Virtue enforces the usual firewalls between multiple users concurrently using a workstation. In addition, its role in Andrew places other responsibilities related to security on it:

—it emulates UNIX semantics for Vice files;
—it ensures that caching is consistent with protection in Vice;
—it allows owners full control over their workstations, without compromising Vice security; and
—it provides user and program interfaces for explicitly using the security mechanisms of Vice.

The next four sections describe these functions in detail.

7.1 UNIX Emulation

Virtue provides strict UNIX protection semantics for local files and a close approximation for Vice files. Each UNIX file has 9 *Mode* bits associated with it. These mode bits are, in effect, a 3-entry access list specifying whether or not the owner of the file, a single specific group of users, and everyone else can read, write, or execute the file.

Venus does the emulation of UNIX protection for Vice files. In addition to the Vice access list check described in Section 6.1 that performs the real enforcement of protection, the three owner bits of the file mode are used to indicate readability, writability or executability. These bits, which now indicate what can be done to the file rather than who can do it, are set and examined by Venus. They are stored and retrieved, but otherwise ignored, by Vice. For directories, the mode bits are completely ignored. The directory listing program, LS, has been modified in Andrew to omit mode bits for directories and show only the owner bits for files. Figure 7 shows an example of a directory listing in Vice.

This combination of an access list mechanism for directories with a UNIX mode bit mechanism on individual files is an evolved strategy. In a prototype of Andrew, the mode bits of a file were derived from the access list of its parent directory and could not be changed by applications. Unfortunately, a few applications, such as version control software, encode state in the mode bits. In addition, our users expressed the need to prevent themselves from accidentally deleting critical files in a directory. The current mechanism provides closer emulation of UNIX and greater functionality, while retaining much of the conceptual simplicity of the original scheme.

Since the group mechanisms of Vice and standard UNIX are incompatible, Venus does not emulate UNIX group protection semantics. Our experience indicates that no real applications have been affected by this. From the point of view of an application, all Vice files belong to a single UNIX group.

7.2 Caching Protection Information

Although ignorant of the Vice group mechanism, Venus caches protection information. When a directory is cached on behalf of a user, Vice supplies rights information for the user and System:AnyUser. Future requests are checked by

```
mozart> ls -l /cmu/itc/satya
total 120
-rwx         1 satya         1385 Jul 24 14:23 3270.keys
d            4 satya         2048 Nov 17  1986 411
-rw-         1 satya         1979 Oct  3  1986 Buildfile
d            2 satya         8192 Sep 17  1986 Mailbox
d            2 satya        12288 Feb  8 10:39 Maillib
d            2 spooler      18432 Aug  2  1986 PrintDir
d            2 satya         2048 Mar  3  1986 Templates
-rw-         1 satya         5219 Jul 20 16:42 articles.frm
-rw-         1 satya         6620 Jul 20 16:36 articles.per
.....................................................................................
.....................................................................................
l            1 satya           38 Jul  2  1986 personal -> /cmu/itc/satya/private/personal
d           19 satya         2048 Jun 29 09:32 pgms
-r--         1 satya          728 Apr  7 13:14 preferences
d            2 satya         2048 Mar 23 14:49 private
d            5 satya         4096 Jul  2 16:53 public
.....................................................................................
.....................................................................................
```

Fig. 7. List of a Vice directory. This is an example of a directory listing in Andrew. For files, the status of the owner mode bits are shown as "r", "w", and "x". These bits are not shown for directories, since mode bits do not apply. Note the use of a symbolic link to obtain a protection status for the file named "personal" that is different from other files in this directory. The file is physically located in the directory "private" which has more restrictive access than the directory shown here.

Venus without contacting Vice. If a different user on that workstation wishes to access the same directory, and the rights for System:AnyUser are inadequate, Venus explicitly obtains the user's rights from Vice. Protection information can be cached for a small number of distinct users on each directory. If there are more users on a workstation, the protection checks will be functionally accurate, but will take longer because of ineffective caching. Vice notifies Venus whenever the protection on a cached directory changes.

Caching interacts with UNIX semantics in a counterintuitive manner. In UNIX, protection failures can only occur when opening a file. In Andrew, a protection failure can occur when closing a file if the protection on one of the directories in its path was changed while the file was open. There is no simple solution to this problem because Vice cannot delegate the responsibility of checking access on store operations. It cannot trust the access check that Venus performs when opening a cached file.

This difference from UNIX semantics affects a number of common UNIX applications that do not expect the close operation to fail, and hence do not check return codes from it. In rare instances the user of such an application may be unaware that one or more files were not stored in Vice because of a protection violation. We do try to inform users of the problem by printing a message on the workstation console. However, using the console as an out-of-band notification mechanism does not help in situations where there is no user to act upon the message. The only robust solution to this insidious failure mode is to modify the applications to check return codes.

7.3 Superuser Privileges

Certain sensitive operational procedures in UNIX can only be performed by the pseudouser "root". Workstation owners need to become root on occasion to perform these procedures. As a result, root is logically equivalent to a group account as discussed in Section 4. A RPC connection on behalf of root provides no knowledge about which actual user it corresponds to.

A further complication is that the initialization of a workstation causes a number of standard processes belonging to root to come into existence automatically. Since there may be no users logged in, Venus may not have tokens with which to make authenticated connections for these processes.[6] We address these problems by treating root specially and granting it the same default access privileges in Vice as System:AnyUser. RPC connections made on behalf of root are unauthenticated and insecure.

The *Setuid* mechanism in UNIX effectively provides amplification of rights [16]. When a file marked setuid is executed, it acquires the access privileges of the owner of the file rather than the user executing the file. The interpretation and enforcement of the setuid property is done by Virtue, but Vice requires authentication tokens for the owner of the program being run setuid. Since the tokens will not be available except in the unlikely case of the owner of the file being logged in to the workstation, Andrew cannot support the setuid mechanism

[6] Automatic logging in of root would require the password to be stored in the clear on workstations, a security risk we were unwilling to assume.

in its general form. However, many useful system utilities on workstations are owned by root and are run setuid. Since root has only System:AnyUser privileges on Vice files and since RPC connections for root do not require tokens, we are able to support setuid in this limited form.

If naively implemented, setuid programs owned by root would make Trojan horses trivial. A user could become root on the workstation, store a Trojan horse program in Vice, and mark it setuid. If this program were run by any other user, it would be able to compromise that user's workstation. To guard against this, we define a special Vice user, "stem." No one can be authenticated as stem, but a system administrator can make stem the owner of a file. When Venus caches a setuid file owned by stem, it translates the owner to root and honors the setuid property. If the file is not owned by stem, the setuid property is ignored.

In our experience this implementation of the setuid mechanism has proven to be a sound compromise between security, UNIX compatibility, and ease of maintenance of system software. Little disruption has been caused by restricting setuid support to root.

7.4 Vice Interface

Virtue provides a number of programs to allow users to use the security mechanisms of Vice. FS is a program to allow users to set and examine Vice access lists. LOGIN, LOG, and SU are modified versions of standard UNIX programs. They prompt for a password, contact the authentication server, obtain tokens and pass them to Venus. A modified version of the UNIX PASSWD program allows users to change their passwords by contacting the authentication server.

For other applications, Virtue provides a library of routines to get, set, and delete tokens stored by Venus. An important user of these routines is the Andrew version of the standard UNIX program RSH that allows a user to execute a program on a remote workstation. Another important user is REM, a program that makes idle workstations available for remote use [23]. In order that the remote site can access Vice files on behalf of the user, both these programs extract tokens from the user's workstation and send them in the clear to the remote Venus. Sending the tokens in the clear is an obvious breach of security, violating the assumptions of Section 3. Yet these programs are popular in our user community! Unfortunately, there is no simple fix to make these programs more secure. To perform the mutual authentication handshake described in Section 5, the local and remote sites would need to share a secret key. No such key exists between an arbitrary pair of Andrew workstations.

There are occasions when a user may wish to voluntarily restrict personal rights, for example, by running a program being debugged in an environment that will not allow it to modify critical files. Virtue allows a user to temporarily disable personal membership in one or more groups, with the group System:Administrators being disabled by default. Disabled groups may be enabled at a later time. At present, a user does not have to be reauthenticated when enabling a group. An additional measure of security would be provided by requiring this.

To implement this temporary disabling of membership, Virtue associates an integer called a *Process Access Group* (*PAG*) with each process. When a process

forks, its child inherits the PAG. Venus associates secure RPC connections to a server with (user, PAG) pairs. Usually all the processes of a user have a single PAG. If a user disables his membership in a group, the process in which the disabling command was issued acquires a new PAG that is distinct from all other PAGs created in this incarnation of UNIX. Each time another server is contacted on behalf of the new (user, PAG) pair, Venus makes a secure RPC connection and requests the server to disable membership in the specified groups. The server constructs a reduced CPS for that connection and uses it on access list checks. PAGs also change when a LOG or SU command is executed.

8. RESOURCE USAGE

The absence of a focal point for allocation of resources makes resource control difficult in a distributed system. Processes in a typical timesharing system are constrained in the rate at which they can consume resources by the CPU scheduling algorithm. No such throttling agent exists in a typical distributed system. Another significant difference is that a process in a timesharing system has to be authenticated before it can consume appreciable amounts of resources. In contrast, each Andrew workstation can be modified to anonymously consume network bandwidth and server CPU cycles.

As discussed in Section 3, Andrew is not designed to be immune to security violations by denial of resources. However, it does provide control over some of the resources. The major resources in Andrew are

—network bandwidth,

—server disk storage and CPU cycles,

—workstation disk storage and CPU cycles.

In the next three sections we examine how Andrew treats these resources.

8.1 Network Bandwidth

Since Andrew does not provide mechanisms to control use of network bandwidth, responsible use of the network is primarily achieved by peer pressure and social mores of the user community. Blatant misuse, such as by flooding with packets, is relatively easy to detect. But it is hard to detect subtle misuse. For example, a malicious user can generate a level of traffic that degrades performance but does not bring useful network activity to a standstill. Or the user can use multiple widely separated public workstations to generate high volumes of traffic. Identifying the user can be particularly difficult because workstations can be modified to generate packets with arbitrary source addresses.

In our experience, network-related problems have not been due to malicious activity. Occasionally, we observe high network utilization and poor file transfer rates on segments of the network that support nonAndrew diskless workstations. The problem has not proved serious enough yet to warrant special attention. In one memorable instance, a bug in the low-level network code on workstations was triggered by a malformed broadcast packet generated by a nonmalicious user during debugging. The bug affected every workstation in the environment and effectively halted all of them.

8.2 Server Usage

Because of the long-term, shared nature of the resource, we felt it important to be able to control disk usage on servers. An Andrew system administrator can specify a storage quota for the Vice files of a user. The quota is actually placed on a *Volume*, an encapsulation of a small subtree of the Vice file space [29]. Quotas can be easily changed by system administrators.

When storing a file on behalf of a user, a server will abort a store operation if the quota is exceeded. This can cause a problem similar to the one described in Section 7.1; an application program that does not check the return codes from a close operation will not report a failure caused by the quota being exceeded. But our users and system personnel consider server disk storage an important enough resource that they have tolerated this problem.

A minor exposure arises from the manner in which electronic mail is implemented in Andrew. Each user has a mailbox directory on which System:AnyUser has insert rights. Mail is delivered by storing each message in its own file in this directory. A malicious user could exhaust the quota of another user by sending large quantities of junk mail. In practice, this has not proved to be a problem.

Although a user cannot execute a program on a server, the user's Venus can consume server CPU cycles in file system operations. Excessive demands on a server are a form of resource denial to other users. At present, Vice does not constrain the amount of server CPU cycles a user can utilize. It could do so, if necessary, since user requests come in on distinct RPC connections.

8.3 Workstation Usage

Andrew does not restrict the amount of space used by local files on workstations. For cached Vice files, Venus employs an LRU algorithm to limit disk usage below a value specified at initialization. The algorithm is not infallible because read and write operations are not intercepted by Venus. It is possible for a program to open a short file and then append a large amount of data, thereby exceeding the cache limit. In practice, this has rarely been a problem.

Since a workstation can be privately owned, it would seem inappropriate for Andrew to constrain the use of its CPU cycles. However, the problem has proved more complex than we anticipated. The primary source of difficulty is the fact that each workstation is a full-fledged UNIX system. Hence it is possible to remotely access one workstation from another via standard UNIX programs such as TELNET and RSH. Since the Vice file space is identical at all workstations, it is particularly easy for a user to use any workstation. Such convenience was, of course, a fundamental motivation for the distributed file system.

Unfortunately, an individual at a workstation perceives the attempt to use its cycles by another user as a security violation. This perception is particularly strong if the first user is at the console of the workstation. Totally disabling the network daemons that allow remote access is not a viable solution for two reasons. First, system personnel sometimes need to remotely access workstations for troubleshooting. Second, an owner may wish to access the workstation from home. Our modem access facilities require the network daemons to be present.

We have evolved a mechanism whereby TELNET access to a workstation can be restricted to a list of users stored in the local file system of that workstation.

This restriction is, however, stronger than what most users desire. When not using the workstation, a user is usually amenable to others using it. It is also unacceptable for public workstations, because every Andrew user should be able to use them. At the present time we do not have a completely satisfactory solution to this resource problem. The REM system, mentioned in Section 7.4, allows a user to specify the conditions that must be satisfied for the workstation to become available for remote use. Although satisfactory to a logged-in user, this approach is harsh on the REM user who is in constant danger of having computation aborted at the remote site. A full-fledged *Butler* mechanism [8] that migrates remote users rather than aborting them would be a more acceptable alternative.

The problem of controlling workstation CPU usage will become acute as Andrew grows. The large pool of idle workstations available for parallel computation, and the development of applications that exploit such parallelism, will make remote use even more attractive in future.

9. ENCRYPTION

Security in Andrew is predicated on the ability of clients and servers to perform encryption for authentication and secure communication. The design and implementation of the encryption algorithm has to satisfy certain properties:

—it must be difficult to break, given the computational resources available to a malicious individual in a typical Andrew environment.
—it must be fast enough that neither the latency perceived by clients nor the throughput of servers be noticeably degraded.
—it must be cheap enough that it does not appreciably increase the cost of a workstation owned by an individual.

Based on considerations of strength and standardization, we have chosen the *Data Encryption Standard (DES)* [20, 34] published by the National Bureau of Standards as the preferred encryption algorithm in Andrew. Since the encryption algorithm is a parameter to our RPC mechanism, it is possible to use other algorithms. We believe, however, that standardizing on DES is appropriate in our environment. This algorithm has been publicly scrutinized for many years, and, although concerns have been expressed about its strength [9], we feel that DES is adequate for the level of security we require.

At the present time the latency for a simple interaction between a client and server is about 20 to 25 milliseconds, and the file transfer rate is about 50 to 70KB/s. We expect these numbers to improve over time as Venus, Vice, and the routers in the network are improved. The fastest software implementation of DES that we are aware of runs at less than 10KB/s on a typical workstation. Software encryption would therefore be an intolerable performance bottleneck in our system; hardware is essential.

It is important to note that Andrew depends on end-to-end encryption where the ends are user-level processes on workstations and Vice servers. Since every connection has a distinct key, the RPC software needs fine-grained and efficient control over the key used to encrypt or decrypt a packet. This implies that link-level encryption devices with fixed or long-term keys are unsuitable.

Although a number of VLSI chips for DES are available [2, 36], integration of such chips into workstation peripherals is not common. A commercially available device for the IBM PC-AT [14, 15] could be used in our IBM RT-PC workstations, but its performance of 50KB/s is barely adequate. We have therefore built a prototype device [7] for the IBM RT-PCs using the AMD 9568 chip. On the basis of the cost and labor of our parts, we estimate that a commercial version of this device, produced in quantity, would cost an end user between 500 and 800 dollars. As perceived by a user-level process, the time to encrypt N bytes using the device is $N * k + C$, where k is 4 microseconds per byte and C is 470 microseconds. The overhead of the device is thus under a millisecond for a small packet and the asymptotic encryption rate is about 200KB/s. We are confident that the device can be redesigned to reduce k in the above expression to about 0.6 microseconds per byte, yielding an asymptotic encryption rate of over 1MB/s. At the present time, we do not have encryption devices for the Sun and Microvax workstations in our environment.

A difficult nontechnical problem is justifying the cost of encryption hardware to management and users. Unlike extra memory, processor speed, or graphics capability, encryption devices do not provide tangible benefits to users. The importance of security is often perceived only after it is too late. At present, encryption hardware is viewed as an expensive frill. We believe, however, that the awareness that encryption is indispensable for security in Andrew will eventually make it possible for every client and server to incorporate a hardware encryption device.

In the interim, while the logistic and economic aspects of obtaining encryption hardware are being addressed, Andrew uses exclusive-or encryption in software. Although it is trivially broken, we felt it worthwhile to use it for two reasons. First, it exercises all paths in our code pertaining to security and allows us to validate our implementation. Second, although a weak algorithm, it does require a user to perform an explicit action to violate security by decrypting data. Merely observing a sensitive packet on the network by accident will not divulge its contents.

10. OTHER SECURITY ISSUES

We now consider two unrelated questions from the viewpoints of security in Andrew:

—how do low-power personal computers access Vice files?
—can diskless workstations be made secure?

Sections 10.1 and 10.2 examine these questions. In focusing only on security, our discussion ignores many broader issues and implementation details.

10.1 PC Server

Many Andrew users also use personal computers such as the IBM PC and Apple Macintosh, and desire Vice access from PCs. PCs in our environment typically have limited amounts of main memory and sometimes lack a local disk. These

characteristics make them substantially different from the class of powerful, resource-rich UNIX workstations for which the design of the Andrew File System is optimized. Rather than compromising the design of the latter to accommodate PCs, we use a surrogate server to interface PCs to Vice. A collection of PCs communicate with the surrogate, called *PCServer*, running on an Andrew workstation. PCServer mediates Vice access from the PC and makes Vice files transparently accessible to the latter.

Communication between a PC and PCServer uses a protocol distinct from that used in the Andrew file system. To perform the 3-way BIND handshake described in Section 5.1, each site running PCServer would need to share a secret key with users at PCs. This could be done using tokens, in a manner analogous to that used by file servers in Vice. But our implementation does not do this. Rather, it supports a weaker form of authentication. The workstation running PCServer also runs an authenticator process called *Guardian*. When a PC user needs to access Vice files, the user supplies his or her Andrew user id and password. These are transmitted to Guardian, logically in the clear, but encrypted with a fixed key. The encryption protects the password against accidental exposure, but not against malicious attacks. Guardian contacts the Andrew authentication server and obtains authentication tokens in a manner identical to LOGIN, as described in Section 5.2. Guardian hands these tokens to Venus and then forks a dedicated UNIX PCServer process on behalf of the user. This process acts on behalf of the PC user and services file requests from the user's PC.

From the point of Venus, it appears as if the PC user had actually logged in at the workstation running PCServer. Enforcement of protection for Vice files is performed exactly as described in Section 6.2. The main security exposure in using PCServer is the information sent in the clear between the PC and Guardian during the establishment of a session. As mentioned earlier, it is technically feasible to fix this problem. But the implementation effort to do this has not been forthcoming.

10.2 Diskless Workstations

Operating workstations without local disks has been shown to be viable and cost-effective [19]. However, the impact of diskless operation on security has been ignored in the literature. To be secure when operating diskless, at least two factors have to be considered. Page traffic has to be encrypted and workstations have to be confident of the identity of their disk servers so that Trojan horses are avoided.

How fast will encryption have to be done to avoid significant performance penalty when running diskless? Cheriton and Zwaenepoel [6] present data from the V kernel on a Sun workstation indicating that it takes about 5 milliseconds plus disk access time to remotely read or write a random 512-byte block of data. These numbers are for file access; but, to a first approximation, we assume that they also hold for page access. We also assume that the server does write-behind, that pages are stored encrypted on the server, and that encryption and decryption take about the same amount of time. Under these conditions, a page fault with replacement of a dirty page would involve a remote page store (5ms), a disk read at the server (20ms for a typical disk), and a remote page fetch (5ms), yielding a

page fault service time of 30 milliseconds.[7] If we require that encryption is to degrade paging performance by no more than 5 percent, it has to be possible to encrypt two 512-byte pages in no more than 1.5 milliseconds. This implies an average encryption rate of about 700KB/s. For the more typical UNIX page size of 4K bytes, an encryption rate in the range of 0.5 to 1MB/s still seems necessary. As described in Section 9, encryption hardware whose performance meets these demands seems feasible, though not readily available.

Authentication is also a difficult problem. To perform an authentication handshake, the client and server need to share a secret key. Where can this key be stored at the client? Embedding it in the ROM containing the boot sequence seems to be the only realistic solution, especially if the workstation has to be able to come up unattended after a power failure. Unfortunately, this violates the goal mentioned in Section 5.2, of not storing long-term authentication information in the clear on workstations. Authentication based on a public key scheme may be a better alternative. In such a scenario, the public keys of legitimate servers would be widely known and could be safely stored in the ROMs of workstations. A server would digitally sign each packet of the boot sequence with its private key, and clients would verify the signature using the public key of that server. Although this scheme only guarantees that the server is genuine, it is likely to be all that is needed for this application.

In fairness, it must be pointed out that Andrew's use of unauthenticated connections to obtain fresh versions of system software after a reboot is also vulnerable to Trojan horse attacks. The problem could, however, be easily alleviated by deferring the update of system software until a user logs in, and then using the user's authenticated connections to perform the update. The integrity of system software on the local disk is also critical, but this is consistent with the assumption, stated in Section 3, that users are responsible for the physical security of the workstations they use.

Although the security problems of diskless workstations are not insurmountable, we know of no real implementations that address them. Concerns regarding security played a small but nontrivial part in our decision to avoid diskless operation in Andrew.

11. RISK ANALYSIS

In this section we briefly consider how security could be subverted in Andrew. Our analysis is not intended to be exhaustive, nor is it a proof of security. Its primary purpose is to summarize the discussions of the preceding sections of this paper. A secondary goal is to illustrate the complexity of applying relatively simple security algorithms to a real distributed environment of substantial scale and diversity.

A fundamental assumption in Andrew is that encryption of sufficient strength and speed is available to Vice and Virtue. Otherwise it is trivial to violate security. For the purposes of this section, we assume that all servers and workstations have such encryption hardware. We also assume that all RPC connections on behalf of users are authenticated and fully encrypted.

[7] If a code page is being evicted the time would be slightly less, that is, 25 ms, since it does not have to be written back to the server.

Low-level network attacks can, at worst, result in denial of service to users. Since RPC packets are encrypted end-to-end, eavesdropping will not reveal useful information. Mutilating RPC packets will not violate security either. Such packets will be rejected by the recipient because RPC sequence numbering information is encrypted, and it is unlikely that a multilated RPC packet will have the correct sequence number when decrypted. For greater confidence, a checksum of the entire packet should be included in the encrypted header.

Breaking of keys by cryptanalytic attacks are much less likely than violations of physical or procedural security. However, one could imagine a malicious individual with patience and considerable computational resources eavesdropping on client-server traffic in Andrew and breaking the key under which the traffic is encrypted. Since a new random session key is generated when a RPC connection is established, breaking that key will only give access to one server. To masquerade as the user, the eavesdropper would have to carefully intersperse fake RPC requests encrypted under the session key. The session key is not adequate to establish connections with other servers.

Greater damage can be done by breaking the key in secret and clear tokens. One way to do this is to break the key used by the authentication server for encrypting secret tokens. This is extremely serious, since all tokens based on that key are compromised. Periodic changing of this key, and careful safeguarding of it, is essential. Another way to break the key in a token pair is to observe a number of BIND requests that involve the same pair of tokens. This is unlikely, because tokens expire after 24 hours, and the number of BIND requests made by a user in that period is not likely to be sufficient to mount a serious key-breaking effort. A compromised token pair allows the miscreant to establish secure RPC connections with the privileges of the victim on any Vice file server. It is not adequate, however, to establish a secure connection to the authentication server.

The most damage is caused when the password of a user, particularly one who is a system administrator, is obtained. However, the password is typically used only once a day when the user is contacting an authentication server for tokens. The standard practice of changing passwords periodically will reduce the total amount of information available for key-breaking.

A well-known mode of attack is via a Trojan horse. Public workstations are particularly susceptible to this. A Trojan LOGIN program on a workstation could compromise the password of every individual who uses that workstation. Concerned sites should insist that users reboot a workstation before using it so as to defeat user-level Trojan horse attacks at login. Further, such sites should ensure that standalone rebooting of a workstation is impossible for normal users. This would defeat the simplest way for malicious users to obtain superuser privileges.

A more subtle way to introduce a Trojan horse is by masquerading as a server that is temporarily down and then handing out fraudulent binaries. During their reboot sequence, workstations fetch new copies of local binaries from Vice over insecure connections. As mentioned in Section 10.2, this problem could be avoided by disabling the automatic update of system software on reboot.

Workstations with multiple logged-in users make a number of other security threats possible. A malicious user with superuser privileges could cause Venus to dump core, examine the dump, and extract the tokens of other logged-in users. Andrew does not provide any special mechanisms to protect against such threats.

As mentioned in Section 3, users of a shared workstation have to trust all individuals who could become superusers on that workstation. A superuser can also read and modify all cache copies of files on the workstation.

Vice is critically dependent on the physical security of its servers and on carefully restricted superuser access on them. For maximum security, servers should disallow TELNET access. Physically secure machine rooms and trustworthy operators are, of course, also essential. A malicious individual with superuser access on a server could read or modify all Vice file data.

Membership in the group System:Administrators has to be carefully guarded. A system administrator can modify any access list in the system, and can therefore read or write any file. The user can also change storage quotas and modify the ownership of files. For increased security, it would be relatively simple to modify Vice to grant System:Administrator privileges only to individuals who, in addition to being authenticated, are logged in at one of a specific set of physically secure workstations.

To keep things in perspective, it should be noted that this section is deliberately negative in tone. Most of the scenarios described here are highly unlikely, and typically involve the violation of the assumptions discussed in Section 3. A site that adheres to those assumptions will find Andrew more secure than any existing distributed system of comparable functionality. Further, in spite of the attention it pays to security, Andrew remains a highly usable system.

12. RELATED WORK

As is often the case in other areas of research, there are many instances where variants of the same basic idea have been independently developed by different groups working on system security. Although Andrew is unique as a system, many of its individual security mechanisms and design decisions resemble those of other systems. We examine the most prominent of these similarities in this section.

Treating the RPC transport protocol as the level of abstraction at which to apply end-to-end authentication and secure transmission measures is a key design decision. The Cedar RPC package [4] was the first to do so; Andrew independently choosing the same approach. More recently, Sun Microsystems has extended its RPC package to support authentication (but not encrypted transmission) [32, 33].

The authentication model used by Kerberos [31] in Project Athena closely resembles Andrew's two-step authentication scheme. A user is required to supply a password only once per login, with Kerberos generating authentication *tickets* for further use. The Kerberos authentication server is replicated, with a single master and multiple read-only slaves. Kerberos guards against replays of service requests by the use of unique *authenticators* supplied by the client with each service request. The use of authenticators requires clocks on servers and clients to be closely synchronized. Andrew does not depend on synchronized clocks. Rather, it uses a connection-based RPC, depending on the encrypted, monotonically increasing sequence numbers in packet headers to guard against replays of service requests.

Physical security of the authentication servers is required in both Kerberos and Andrew, since they store passwords in the clear. In contrast, the key distribution mechanism used by Sun [32, 33] avoids storing passwords in the clear. The common DES key needed for mutual authentication is obtained from information stored in a publicly readable database. Stored in this database for each user and server is a pair of keys suitable for public key encryption. One key of the pair is stored in the clear, while the other is stored encrypted with the login password of the user. Any two entities registered in the database can deduce a unique private key for mutual authentication.

Apollo has recently extended the UNIX protection mechanism on their workstations with access lists [11]. But the details of the approach differ significantly from those of Andrew. Apollo access lists are optional, and can be associated with individual files and directories. A set of precedence rules determine the rights accruing to a user if multiple entries of an access list apply to him. This is in contrast to Andrew, where the user obtains the union of these rights.

The use of a hierarchical protection domain in Andrew is inherited from the CMU-CFS file system [1]; Grapevine [3] also used a similar scheme.

13. CHANGES SINCE SNAPSHOT

As mentioned in Section 1, the details presented in the preceding sections of this paper pertain to a snapshot of Andrew at one point in time. Andrew was modified in an incremental manner from the date of that snapshot, November 1986, until the summer of 1988.

Most of these modifications were improvements to existing functionality. The protection database was changed so that its index was stored internally, rather than in a separate file. This eliminated the occasional inconsistencies between index and data that used to occur when propagating protection domain information. The *RPC2* remote procedure call mechanism, described in Section 5.1, was replaced by *R*, a similar RPC mechanism that was more parsimonious in its use of memory. The replacement enabled Venus to run on workstations with very limited physical memory. Although there were differences in the details, the authentication handshakes used by RPC2 and R were conceptually similar.

A significant enhancement to functionality was the addition of support for multiple *Cells* [38]. A cell corresponds to a completely autonomous Andrew system, with its own protection domain, authentication, file servers, and system administrators. A federation of cells can cooperate in presenting users with a uniform, seamless file-name space. Yet, for smooth and efficient operation, cells allow administrative responsibility to be delegated along lines that parallel institutional boundaries.

Although the presence of multiple protection domains complicates the security mechanisms in Andrew, Venus hides much of the complexity from users. For example, authentication tokens issued in a cell are valid for use within that cell only. Venus maintains a collection of tokens, one pair for each cell to which the user has been authenticated. When establishing a secure connection to a Vice server, it uses the tokens appropriate to the cell in which the server is located. A user who has not been authenticated to that cell gets System:AnyUser privileges

in it. A user is aware of the existence of cells only in the beginning, when directing authentication requests to individual cells using the LOG program described earlier. In a small number of cases, application programs have to be modified to make cells transparent. For example, a user registered in multiple cells may have Vice ids that are different in each cell. An application such as the directory listing program, LS, that translates ids to user names has to be modified to display names correctly.

Since the summer of 1988, a major reimplementation of the Andrew file system has been under way. This will result in a new version, AFS 3.0 [30]. The goals of this reimplementation are improved performance, ability to operate over wide-area networks, improved operability and system administration, and better standardization of components. Three major changes pertain to security. First, the new RPC mechanism, *Rx*, provides support for a variety of security modules. Different instances of Rx can use different authentication and encryption mechanisms. Second, in the interests of standardization, Kerberos from Project Athena is being used for authentication. Third, a *protection server* is being implemented. This will allow users to create and manipulate groups themselves, rather than depending on system administrators to perform this function.

14. CONCLUSION

Throughout the evolution of Andrew, the underlying model of security has remained unchanged. A small collection of trusted servers jointly provide a secure storage repository for users at a large number of workstations. The security of the entire system is not contingent upon the integrity of these workstations or of the network. Loss of integrity of a workstation can, at worst, compromise the security of files accessible to those who use the workstation. This is a substantially higher level of security than that offered by most contemporary distributed systems. Our experience with Andrew gives us confidence that this level of security can be attained without significant loss of usability or performance, even at large scale.

Security will be a more serious issue in the future. Although much theoretical research has been done in this area, applying those principles to real systems is difficult. The factors contributing to complexity include the many levels of abstraction spanned, the need for compatibility, and the existence of numerous minutiae that have to be correctly addressed. In spite of this, we are convinced that the distributed systems of the future will have to pay greater attention to security if they are to remain viable.

ACKNOWLEDGMENTS

Although this paper has a single author, many individuals deserve credit for their valuable contributions to the work reported here.

The security architecture of Andrew was developed by the File System Group of the Information Technology Center at Carnegie Mellon University. The membership of this group over time has included John Howard, Michael Kazar, David King, Sherri Menees Nichols, David Nichols, M. Satyanarayanan, Robert Sidebotham, Michael West, and Edward Zayas. Preliminary design work on

security in Andrew was done by David Gifford, M. Satyanarayanan, and Alfred Spector.

Many of the mechanisms described in this paper were implemented by M. Satyanarayanan. David Nichols and Michael Kazar contributed to the mechanisms in Virtue, while Michael West contributed to those in Vice. The security mechanisms that support PC Server were built by Larry Raper and Jonathan Rosenberg. Paul Crumley designed and implemented the encryption hardware for RT-PCs.

James Kistler, James Morris, Jonathan Rosenberg, Robert Sansom, Ellen Siegel, Doug Tygar, and the journal referees provided many useful comments on this paper.

REFERENCES

1. ACCETTA, M. J., ROBERTSON, G. G., SATYANARAYANAN, M., AND THOMPSON, M. The design of a network-based central file system. Tech. Rep. CMU-CS-80-134, Dept. of Computer Science, Carnegie Mellon Univ., Pittsburgh, Pa., Aug. 1980.
2. ADVANCED MICRO DEVICES. *MOS Microprocessors and Peripherals*, 1985.
3. BIRRELL, A. D., LEVIN, R., NEEDHAM, R. M., AND SCHROEDER, M. D. Grapevine: An exercise in distributed computing. *Commun. ACM 25*, 4 (Apr. 1982), 260–273.
4. BIRRELL, A. D. Secure communication using remote procedure calls. *ACM Trans. Comput. Syst. 3*, 1 (Feb. 1985), 1–14.
5. BURROWS, M. L., ABADI, M., AND NEEDHAM, R. N. A logic of authentication. Tech. Rep. 39, Digital Equipment Corporation, Systems Research Center, Palo Alto, Calif., Feb. 1989.
6. CHERITON, D. R., AND ZWAENEPOEL, W. The distributed V kernel and its performance for diskless workstations. In *Proceedings of the 9th Symposium on Operating System Principles* (Bretton Woods, N.H., Oct. 11–13, 1983). ACM, New York, 1983 pp. 129–140.
7. CRUMLEY, P. *TRADMYBD: Data Encryption Adapter Technical Reference Manual and Programmers' Guide, Version* 0.20. Tech. Rep. CMU-ITC-059, Information Technology Center, Carnegie Mellon Univ., Pittsburgh, Pa., Dec. 1986.
8. DANNENBERG, R. B. Resource sharing in a network of personal computers. Ph.D. dissertation, Dept. of Computer Science, Carnegie Mellon Univ., 1982.
9. DIFFIE, W., AND HELLMAN, M. E. Privacy and authentication: An introduction to cryptography. *Proc. IEEE 67*, 3 (Mar. 1979), 397–427.
10. DoD. *Trusted Computer System Evaluation Criteria.* CSC-STD-001-83, Dept. of Defense, Computer Security Center, 1985.
11. FERNANDEZ, G., AND ALLEN, L. Extending the UNIX protection model with access control lists. In *Usenix Conference Proceedings* (Summer, 1988).
12. GRAMPP, F. T., AND MORRIS, R. H. Unix operating system security. *Bell Lab. Tech. J. 63*, 8 (Oct. 1984), 1649–1672.
13. HOWARD, J. H., KAZAR, M. L., MENEES, S. G., NICHOLS, D. A., SATYANARAYANAN, M., SIDEBOTHAM, R. N., AND WEST, M. J. Scale and performance in a distributed file system. *ACM Trans. Comput. Syst. 6*, 1 (Feb. 1988), 51–81.
14. IBM. *IBM 4700 Personal Computer Financial Security Adapter: Guide to Operations.* No. 6024361, IBM Corp., 1985.
15. IBM. *IBM 4700 Personal Computer Financial Security Adapter: Microcode Users Guide.* No. 6024362, IBM Corp., 1985.
16. JONES, A. K. Protection in programmed systems. Ph.D. dissertation, Dept. of Computer Science, Carnegie Mellon Univ., Pittsburgh, Pa., 1973.
17. JONES, A. K., AND WULF, W. A. Towards the design of secure systems. *Softw. Pract. Exper. 5* (1975), 321–336.
18. LAMPSON, B. W. A note on the confinement problem. *Commun. ACM 16*, 10 (Oct. 1973), 613–614.
19. LAZOWSKA, E. D., ZAHORJAN, J., CHERITON, D. R., AND ZWAENEPOEL, W. File access performance of diskless workstations. *ACM Trans. Comput. Syst. 4*, 3 (Aug. 1986), 238–268.

20. MEYER, C. H., AND MATYAS, S. M. *Cryptography: A New Dimension in Computer Data Security.* John Wiley, New York, 1982.
21. MORRIS, J. H., SATYANARAYANAN, M., CONNER, M. H., HOWARD, J. H., ROSENTHAL, D. S., AND SMITH, F. D. Andrew: A distributed personal computing environment. *Commun. ACM 29,* 3 (Mar. 1986), 184–201.
22. NEEDHAM, R. M., AND SCHROEDER, M. D. Using encryption for authentication in large networks of computers. *Commun. ACM 21,* 12 (Dec. 1978), 993–998.
23. NICHOLS, D. A. Using idle workstations in a shared computing environment. In *Proceedings of the 11th ACM Symposium on Operating Systems Principles* (Nov. 8–11, 1987). ACM, New York, 1987, pp. 5–12.
24. RABIN, M. O., AND TYGAR, J. D. An integrated toolkit for operating system security. Tech. Rep. TR-05-87, Aiken Computation Lab., Harvard Univ., Cambridge, Mass., May 1987.
25. SALTZER, J. H. Protection and the control of information sharing in Multics. *Commun. ACM 17,* 7 (July 1974), 388–402.
26. SATYANARAYANAN, M. Users, groups and access lists: An implementor's guide. Tech. Rep. CMU-ITC-84-005, Information Technology Center, Carnegie Mellon Univ., Pittsburgh, Pa., Aug. 1984.
27. SATYANARAYANAN, M., HOWARD, J. H., NICHOLS, D. N., SIDEBOTHAM, R. N., SPECTOR, A. Z., AND WEST, M. J. The ITC distributed file system: Principles and design. In *Proceedings of the 10th ACM Symposium on Operating System Principles* (Dec. 1–4, 1985). ACM, New York, 1985, pp. 35–50.
28. SATYANARAYANAN, M. *RPC2 User Manual.* Tech. Rep. CMU-ITC-84-038, Information Technology Center, Carnegie Mellon Univ., Pittsburgh, Pa., 1986 (revised).
29. SIDEBOTHAM, R. N. Volumes: The Andrew file system data structuring primitive. In *European Unix User Group Conference Proceedings* (Aug. 1986). Also available as Tech. Rep. CMU-ITC-053, Information Technology Center, Carnegie Mellon Univ., Pittsburgh, Pa., 1986.
30. SPECTOR, A. Z., AND KAZAR, M. L. Wide area file service and the AFS experimental system. *Unix Rev. 7,* 3 (Mar. 1989).
31. STEINER, J. G., NEUMAN, C., AND SCHILLER, J. I. Kerberos: An authentication service for open network systems. In *Usenix Conference Proceedings* (Winter, 1988).
32. TAYLOR, B. Secure networking in the Sun environment. In *Usenix Conference Proceedings* (Atlanta, Ga., Summer, 1986).
33. TAYLOR, B. A framework for network security. *SunTechnology 1,* 2 (Spring 1988).
34. U.S. DEPARTMENT OF COMMERCE, N.B.S., *Data Encryption Standard.* 1977. Federal Information Processing Standards Publication, FIPS PUB 46.
35. VOYDOCK, V. L., AND KENT, S. T. Security mechanisms in high-level network protocols. *ACM Comput. Surv. 15,* 2 (June 1983), 135–171.
36. WESTERN DIGITAL CORP. *Data Communication Products Handbook,* 1985.
37. WULF, W. A., LEVIN, R., AND HARBISON, S. P. *HYDRA/C.mmp: An Experimental Computer System.* McGraw-Hill, New York, 1981.
38. ZAYAS, E. R. Administrative cells: Proposal for cooperative Andrew file systems. Tech. Rep. CMU-ITC-060, Information Technology Center, Carnegie Mellon Univ., Pittsburgh, Pa., June, 1987.

Received November 1987; revised January 1989; accepted April 1989