The following paper was originally published in the
Proceedings of the Twelfth Systems Administration Conference (LISA '98)
Boston, Massachusetts, December 6-11, 1998

Single Sign-On and the System Administrator

Michael Fleming Grubb and Rob Carter
Duke University

For more information about USENIX Association contact:

1. Phone: 510 528-8649
2. FAX: 510 548-5738
3. Email: office@usenix.org
4. WWW URL: http://www.usenix.org

Single Sign-On and the System Administrator

Michael Fleming Grubb and Rob Carter – Duke University

ABSTRACT

Large organizations are increasingly shifting critical computing operations from traditional host-based application platforms to network-distributed, client-server platforms. The resulting proliferation of disparate systems poses problems for end-users, who must frequently track multiple electronic identities across different systems, as well as for system administrators, who must manage security and access for those systems. Single sign-on mechanisms have become increasingly important in solving these problems. System administrators who are not already being pressured to provide single sign-on solutions can expect to be in the near future. Duke University has recently embarked on an enterprise-wide single sign-on project. This paper discusses the various factors involved in the decision to deploy a single sign-on solution, reviews a variety of available approaches to the problem of electronic identity proliferation, and documents Duke's research and findings to date.

Introduction

The number of mission-critical computing platforms in today's enterprise is exploding. Applications that were once housed on ''big iron'' (MVS mainframes, enterprise-class Unix servers, etc.) are increasingly replaced with network-distributed client-server applications running across large numbers of smaller systems. Concurrently, the number of end users associated with these systems has increased dramatically in recent years. Together, these changes have resulted in a sizable increase in the number of disparate systems which end users interact with on a regular basis and which system administrators manage.

The forces driving these changes are well-known. Increased functionality and faster response time for end-users, increased demand for access to organizational information, decreased total cost of ownership, and Y2K considerations have all contributed to the explosion in network-based systems. While network-distributed applications have obvious advantages for both end-users and system administrators, their growth poses new problems for authentication, auditing, and security management.

In the traditional, host-based application environment, authentication was simplistic. Users were authenticated by the host operating system as they presented for entry into the system, and were then granted access to applications and data based on their locally-authenticated identities. Multiple applications co-located on a single large host system would share authentication information through the host's operating system, and access to the host system could be managed cheaply and centrally.

Audit trails could also be maintained centrally, and because all applications shared a common, operating system-specific authentication mechanism, could easily be correlated between applications. Each individual would appear with a single identity across all applications on the central host. Therefore, audit records pertaining to an individual's actions within one application housed on the central host could easily be correlated with records pertaining to other actions taken by that same individual. Similarly, security and access management was straightforward, amounting in most cases to little more than maintaining a single user database and various application-specific authorization tables.

With the advent of network-distributed applications and the accompanying increase in the number of disparate enterprise systems, authentication and related issues have become more complex. Each different application or service may require separate authentication for its users. Users of multiple applications or systems may be confronted with a maddening list of disparate electronic identities and authenticators to remember. End users may be required to authenticate themselves multiple times during a single work session, and may have to remember and maintain a daunting number of login ids and passwords.

From the standpoint of the system administrator, the situation is no less disturbing. In the more distributed world, administrators must maintain authentication information across multiple platforms by creating, issuing, and deleting users' authentication information in multiple different contexts. While automated account management systems and distributed management tools (e.g., Tivoli, CA/Unicenter) can reduce the per-user effort involved in managing user identities across multiple systems, they are often as difficult to manage and maintain as the systems they support.

Further, administrators who manage electronic security for their organizations must often go to great

lengths to ensure that audit trails from disparate systems, regardless of whether they share authentication information with one another, can be reconciled. Actions performed by one individual across a number of different systems may be difficult to correlate with that single individual if the systems involved do not agree as to the individual's authenticated identity.

Additionally, the proliferation of electronic identities has social ramifications that can be devastating to overall system security. Presented with ten or more different authenticators for separate applications and systems, many end-users will resort to using insecure but easily remembered passwords, or will resort to recording their authentication information in an insecure fashion. At Duke, the authors have frequently run across this phenomenon in the form of users (and in some cases, system administrators) taping lists of their various logins and passwords to monitors in their offices. Enforcing any reasonable security policy in such an environment can be well-nigh impossible.

As end-users become increasingly frustrated by these issues, and as organizational leaders grow in their awareness of the serious security ramifications of electronic identity proliferation, pressure increases on systems administrators to provide technical solutions to these problems. Typically, this pressure results in the demand for a ''single sign-on solution.''

Single sign-on is something of a holy grail for large organizations. Everyone seems to want it, many people claim to have it for sale, but no one seems to agree as to what, exactly, it is. To many, ''single sign-on'' means that each user in the enterprise has only one userid and associated password. To others, ''single sign-on'' means that wherever a user logs in, he is presented with an interface and application set that is specifically tailored to him and which follows him throughout the enterprise. Another interpretation, favored by the authors, is that ''single sign-on'' *per se* is the goal of presenting the end user with only one authentication challenge during a single work session.

At Duke, the demand for a ''single sign-on solution'' started in earnest as the University began work on a number of parallel enterprise-wide computing projects. Payroll, human resources, student information systems, and purchasing were all targeted for migration off the institutional MVS mainframe and onto a set of distributed Unix-based systems. As the software engineers working on the deployment of these new systems began to consider work-flow patterns and information sharing requirements, and as users began to realistically consider the effect these new systems would have on their day to day activities, they realized the potential pitfalls in managing user information across such widely-distributed systems.

The authors, as lead system administrators for the arm of the university charged with the design and maintenance of institutional computing infrastructure, were approached with a deceptively simple question: How should the University go about providing a ''single sign-on'' for the many and varied enterprise-wide applications in use and soon to be deployed on campus? A review committee, chaired by the authors, was formed and charged with answering this question. What was originally expected to be a three-month review process followed by rapid deployment of a complete solution has since become a 9-month-long investigation culminating in a recommended institutional strategy for user authentication which, we hope, will limit but not eliminate the proliferation of electronic authenticators across the enterprise. With this paper, we hope to offer other systems administrators the benefit of experience thus far at Duke in designing and implementing a comprehensive single sign-on solution.

Single Sign-On: One Password, Two Flavors

Before designing a single sign-on solution, a system administrator must first determine precisely what ''single sign-on'' means in her local environment. Is the demand for a single sign-on solution being driven by a need for enhanced security and auditability, or is it more an outgrowth of end-user frustration with the proliferation of electronic identities? To what extent are existing systems, many of which may already house multiple authenticators for individual users, to be participants in a single sign-on solution? Is the user population highly mobile, or do individual users work from fixed locations, and to what extent do individual users need simultaneous access to multiple different applications or systems? Answers to these and other basic questions will help focus efforts on solutions appropriate to local environments.

In particular, it is critical that organizations investigating single sign-on solutions decide whether their need is for a method by which to reduce the number of authenticators each end-user must remember and maintain, or whether their need is for a method by which to reduce the number of authentication operations a given user must perform in the course of a single work session. In the former case, the ultimate goal is the creation of a Single Authentication Realm, or SAR, where each user has only one authenticator (userid and password). In the latter case, the ultimate goal is the creation of an actual Single Sign-On, or SSO, mechanism, where each user authenticates only once during each work session. In the SAR case, individual users may be required to authenticate multiple times during a given work session, typically once for each disparate application or system accessed. In the SSO case, each user performs at most one authentication operation during each work session, although multiple authentication operations may be performed on behalf of the user by SSO software during the course of the user's work.

Building an enterprise-wide SAR is not a necessary prerequisite to an SSO solution, but in many cases it may be sufficient to meet an organization's

needs. A SAR can eliminate many of the security risks associated with electronic identity proliferation. With only one set of authentication information to remember and maintain, users are less likely to practice poor password hygiene and are more likely to protect their authenticator information. A SAR can also eliminate many of the system administration problems posed by userid proliferation. As each user has only one authenticated identity, reconciling audit trails across systems participating in the SAR becomes as easy as reconciling audit trails across multiple host-based applications on a single host, and user identity management (creating and removing authentication information for individual users) is reduced to managing data associated with the SAR. A SAR can also significantly reduce the overall cost of managing access to critical systems by reducing the number of times individual users must be initially certified (that is, the number of times individuals' actual identities must be ascertained and recorded in order to validate new authentication identities issued to them). With a functioning SAR, each individual need only be identified by system staff once (to authenticate their identity within the SAR at its inception), so strong certification policies can be enforced with less time investment by both end-users and system administrators.

A SAR, however, may not be sufficient to satisfy the demands of an organization's user population. While each user may have only one authenticator (e.g., a login/password pair) to remember and maintain, users may still be annoyed by the need to repeatedly enter authentication information during a single work session. At Duke, this has been of particular concern to the health-care community associated with the University's Medical Center. Physicians, nurses, and other health-care providers are understandably concerned about the time they may be required to spend repeatedly identifying themselves to different systems during a single session. In life or death situations, the few seconds a physician spends re-authenticating during a critical work session may make a real difference in the outcome of his or her patient's treatment.

Depending on exactly how it is implemented, a SAR may also increase the exposure of secure authentication information on possibly insecure networks, resulting in an overall *decrease* in system security. In essence, although each user need only remember one combination to open any organizational safe, the likelihood of that combination being compromised by a determined safe-cracker bent on watching the user's day-to-day activities may be increased under some circumstances.

Addressing these latter issues usually involves the deployment of a full SSO solution. This may take various forms: a carefully-designed SAR issuing reusable authentication credentials honored by all participating systems and services, a redesigned set of endpoint systems and applications built specifically to rely on a third-party authentication system to identify users, a separate SSO application designed to proxy authentication operations for previously-authenticated users, or some mixture of all three. In addition to the advantages offered by its underlying (or implicit) SAR, an SSO solution can increase overall user satisfaction with computing systems, and can reduce the frequency and extent of a system's security exposure.

Depending on the specific SSO approach chosen, however, deploying and maintaining an SSO solution can be both labor-intensive for the system administrator and expensive. Maintaining the SAR and/or SSO infrastructure itself may in some circumstances become a significant drain on system administrators. Developing and maintaining interfaces between a given SSO mechanism and legacy applications or application environments may also be difficult or impossible. Additionally, cost considerations may limit the extent to which an SSO mechanism can be deployed enterprise-wide.

Application-Independent Authentication vs. Application-Dependent Authorization

It is important to note that both SAR and SSO solutions pertain to *authentication*, rather than *authorization*. Authentication, for purposes of this discussion, is the process by which the unique identity of an individual user is determined and represented in electronic form. Authorization, on the other hand, is the process by which a particular authenticated individual is identified as authorized to access a particular service or datum. As the authors found in early discussions at Duke with interested parties, authentication and authorization are frequently confused with one another, and as the authors also found in early discussions at Duke, authorization issues are frequently much more political than authentication issues.

Authentication is by its nature an application-independent process because an individual's identity is unique to the individual, regardless of what role the individual is fulfilling or what operation the individual is attempting to perform. Authorization is by contrast an application-dependent process because different applications may need to control access within their individual domains quite differently. User authorization poses its own unique set of challenges for both application programmers and system administrators, none of which are directly addressed by SAR or SSO solutions.

Authentication and authorization are, however, closely interrelated. In order to make appropriate authorization decisions, applications must among other things have access to reliable authentication information. An application which cannot uniquely identify its user cannot hope to make appropriate authorization decisions. Ideally, a SAR can provide the basis for building a cross-platform authorization infrastructure within an organization. While authorization mechanisms are outside the scope of this paper, it

should be noted that strong authentication mechanisms can play a part in supporting strong authorization mechanisms.

Options for Establishing a SAR

Once a decision has been made to implement a Single Authentication Realm, the system administrator may be tasked with recommending a particular authentication solution for use within the SAR. There are a variety of authentication options, each with its own strengths and weaknesses. Depending on organization-specific needs, any or all of the available options may need to be investigated or used in the development of an enterprise-wide SAR.

Policy-Based SAR: Solution by Fiat

Perhaps the simplest, although certainly neither the most robust nor the most functional solution the authors have heard of for building a SAR for a large enterprise is to dictate by organizational policy that each user shall use one userid and password for each system and application. Effectively, the organization achieves a SAR by fiat; institutional policy forbids the proliferation of authenticators.

In the authors' experience, such restrictive dicta are virtually impossible to enforce in any but the smallest and most tightly-managed organizations. As the number of users increases, and particularly as there are more system administrators who must cooperate with such a policy in order to make it enforceable, the labor required to enforce the policy increases beyond manageable limits.

Further, it is impossible to enforce such a restrictive policy across disparate systems without sacrificing password security; either those administrators responsible for enforcement of the policy must have access to users' individual authenticator information, or systems must somehow compare authenticators on a regular basis to identify violations of the policy.

Moreover, successful implementation of such a restrictive policy could increase the overall security vulnerability of systems managed under the policy by increasing the number of points from which the security of users' authenticators are vulnerable to attack. Because all cooperating systems must replicate users' authentication information locally, the security of all cooperating systems is limited by the security of the most loosely-managed system in the group. Also, since there is no single location from which all possibly-compromised authenticators owned by a particular individual can be modified or revoked, it becomes time consuming and expensive for system administrators to address emergent security breaches, and users run the risk of neglecting to update one or more compromised authenticators, leaving themselves open to continued attack.

Despite its weaknesses, this approach *can* be made to work under certain circumstances. A similar approach has been in use within the Duke University Medical Center for a number of years. Upon arrival, each employee of the Medical Center is assigned a DEMPO (Duke Electronic Mail Post Office) id which is guaranteed unique within the set of DEMPO ids, along with a DEMPO password. System administrators throughout the Medical Center are required, by policy, to use DEMPO ids in creating accounts on their systems.

When the policy was instituted, in the late 1980s, the Medical Center had only a few multi-user systems, and still fewer system administrators. The majority of computing within the Medical Center was still centralized on a large MVS mainframe, and the majority of departmental computing platforms within the Medical Center were managed by a handful of corporate system administrators. Along with the policy, the Medical Center established an electronic mail service based on the PMDF product from Innosoft, and used a PMDF mail gateway to provide the appearance of a unified electronic mail system using DEMPO ids as electronic mail aliases. This unified electronic mail service made the DEMPO id mandate somewhat more palatable to end-users than it might otherwise have been, and given the limited number of administrators involved, the project was an initial success. Once the DEMPO id policy had become a part of the Medical Center corporate culture, its success as an organizational policy was a *fait accompli*.

A variety of complications have arisen over the years as a result of this policy. For example, DEMPO ids are assigned to end-users based on their surnames at the time of their hiring. Since many individuals change their surnames as a result of changes in marital status, adoption, etc., there are frequent requests to change users' DEMPO ids to match their new surnames, requests which cannot be honored under existing policy. Further, enforcement of the DEMPO id policy has been impossible in an environment in which individual principal investigators have almost complete control over their grant-funded operations, and each failure in enforcement leads to increased friction among those system administrators who are forced to follow the stated policy. Although the policy has been in effect within the Medical Center for a number of years, it continues to be the cause of repeated complaints by end-users and system administrators alike.

Unfortunately, this policy-based approach may often be the first one put forward by managers and administrators eager to arrive at a ''quick fix'' for the problem of identity proliferation. This method may, at first, seem like a low-cost, high-yield solution to what could otherwise be an expensive and complex problem to solve. At Duke, this was the case; it was originally suggested that enforcing a strict policy across systems might provide a simple way to control identity proliferation at the institution. We strongly discourage this

approach, which was ultimately discarded at Duke in favor of other, more feature-rich options.

Network-Based Authentication Mechanisms: A Better Approach

A more sound approach to the creation of a SAR is the use of a network-based authentication service of some kind. A number of different services are available to the system administrator, ranging in complexity from simple network-distributed databases to complex third-party authentication protocols based on various encryption technologies. Each of these approaches has its own strengths and weaknesses, and any or all of them may reasonably be viewed as candidates for partial or complete SAR solutions, depending on the specific requirements of an organization.

The simplest and most widely-utilized network-based solutions are network-distributed databases. By providing network-distributed access to a single authentication database, multiple cooperating systems can share users' authentication information, yielding a SAR. Some of the most common implementations of this approach are little more than network extensions of the well-known Unix *passwd*(4) table mechanism.

Hesiod: Authentication via DNS

One such approach, Hesiod, is a mechanism for distributing passwd table information (or virtually any arbitrary textual information) across a network using extensions to the traditional domain name service (DNS). Developed in the late 1980s at MIT, Hesiod builds on what was and still is a well-supported standard for network-based information distribution. In the Hesiod approach, authentication information traditionally stored in a local Unix passwd table (userids, encrypted passwords, etc.) is stored in extended DNS records and made available via extensions to the normal domain name resolution protocol (cf. RFCs 1034 & 1035). Hesiod-capable DNS servers support, in addition to the traditional DNS records of class ''IN,'' records of class ''HS.'' Class ''HS'' records may include records of type ''TXT,'' containing arbitrary text strings indexed by other arbitrary text strings.

For a number of years, Digital Equipment Corporation's Ultrix operating system provided native support for Hesiod as a means of distributing Unix passwd table information. The DEC implementation of Hesiod was in use for some time within Duke's public Unix computing facilities, and presents a reasonable example of the Hesiod approach.

In this implementation, additional Hesiod pseudo-domains are constructed containing class ''HS,'' type ''TXT'' records carrying user information in the standard Unix passwd and group table formats. The primary DNS server for a given domain, ''team.foo.org'' for example, publishes not only the authoritative ''team.foo.org'' DNS information, but also authoritative Hesiod information for the pseudo-domains ''passwd.team.foo.org'' and ''group.team.foo.org.'' These pseudo-domain tables contain standard passwd and group table information. For example, the ''passwd.team.foo.org'' domain table might include records of the form show in Figure 1 that support access to user passwd table entries indexed on both login id and uid number.

This information, along with standard DNS information, is then made available across a network via an extended version of the standard name resolution protocol. An extension to the standard DNS API is made available through Hesiod-aware replacements for the standard resolver routines (commonly, a single ''hes_resolve()'' routine and a single ''hes_error()'' routine) to allow applications to search for authentication information in the form of passwd table entries. Hesiod queries for records pertaining to ''rgc.passwd.team.foo.org'' or ''103.passwd.team.foo.org'' would, in the example above, return passwd table information for the user ''His Nobs.''

In the DEC implementation of Hesiod, this ability to index individual passwd table entries within a Hesiod server's primary Hesiod tables on multiple keys was used to provide native support within the Ultrix operating system for Hesiod as a primary authentication mechanism. Under Ultrix, Hesiod-aware versions of the standard getpwnam(), getpwuid(), and getpwent() library calls were made available which would, depending on the configuration of a given system, return information retrieved via the Hesiod resolver API. Thus, a homogeneous network of Ultrix machines could achieve the benefits of a SAR through direct use of Hesiod.

Hesiod exhibits a number of strengths. Having been developed as part of the MIT Athena project, source code to implementations of Hesiod is freely available, and having been implemented as an extension to a well-known and well-standardized protocol, Hesiod boasts a level of standards compliance few similar systems achieve. Hesiod tables are relatively easy to manage, being structured similarly to regular DNS tables, and can be distributed across multiple Hesiod servers via the same zone transfer mechanism used to synchronize DNS tables between primary and secondary domain name servers. As is the case with traditional DNS information, Hesiod information can be published by multiple cooperating servers, some of which may act as ''caching'' servers to enhance Hesiod look-up performance across wide-area networks.

```
rgc  HS  TXT  "rgc:PASSWORDCRYPT:103:4:His Nobs:/home/rgc:/bin/bash"
103  HS  TXT  "rgc:PASSWORDCRYPT:103:4:His Nobs:/home/rgc:/bin/bash"
```

Figure 1: Domain table records.

Hesiod is not, however, a perfect solution to the SAR problem. To date, the authors are only aware of one major vendor providing native operating system support for Hesiod as a primary authentication mechanism (Digital Equipment Corporation). Although it is possible to use the open Hesiod API to modify any arbitrary operating system or application to support a Hesiod-based SAR, retrofitting Hesiod support into existing applications (particularly vended applications) can be difficult.

Additionally, Hesiod does not in itself provide any mechanism for securing information stored in Hesiod tables. Not only are Hesiod tables stored in plain text on Hesiod servers, but Hesiod information is also transported over the network in plain text. Further, Hesiod servers do not necessarily enforce any limits on what client machines can gain access to Hesiod information, although recent versions of the BIND name server can be used to address that problem. These properties make Hesiod unsuitable for the distribution of secure information, and depending on the security requirements of specific organizations, unsuitable as the basis for a SAR.

NIS (YP): Authentication Databases via RPCs

Another network-based authentication database approach to developing a SAR is the use of Sun's Yellow Pages (YP) or Network Information Nameservice. Developed by Sun Microsystems under the auspices of the company's ONC development project, NIS (formerly known as Yellow Pages) provides much the same functionality as Hesiod, but uses a completely different network infrastructure. Although NIS was originally developed as a Sun Microsystems initiative, the subsequent opening of Sun's ONC RPC interface has led to a number of different vendors offering support for the mechanism.

In the NIS environment, as in the Hesiod environment, groups of cooperating machines are referred to as domains. Although they frequently coincide with DNS domains, NIS domains are completely orthogonal to DNS domains. Machines in multiple different DNS domains can, in theory, be members of the same NIS domain, and vice-versa.

Like Hesiod, NIS provides a mechanism for distributing arbitrary database tables (termed ''maps'') from a single set of database servers to a number of clients across a network. Also as is the case with Hesiod, the database tables distributed by NIS can include authentication information, typically stored in the standard Unix passwd and group table format.

NIS, however, uses a completely different mechanism to actually distribute information across the network. Whereas Hesiod relies on the pre-existing DNS standard for network extensibility, NIS relies on Sun's ONC RPC mechanism. In the NIS scenario, client machines within a given NIS domain use any of a variety of NIS-specific RPC calls to perform search, retrieval, and update operations on NIS data tables stored on NIS servers. NIS clients are directed to their respective domains' NIS servers via a process of client binding. NIS client machines are intrinsically aware of the NIS servers in their environment, and direct any required RPC calls to their respective servers.

When used as the basis for a SAR, NIS takes the place of the traditional Unix passwd and group table mechanism, much as Hesiod does in the earlier example. NIS client systems are typically equipped with NIS-aware versions of the standard getpwnam(), getpwent(), and getpwuid() routines, allowing the use of NIS for network-based access to authentication information in a manner that is transparent to applications written to use native Unix authentication mechanisms.

As is the case with Hesiod, there may be more than one NIS server configured for a given NIS domain. In such cases, one NIS server acts as the master, processing all updates and periodically conveying new copies of its master tables to the other servers, which act as slaves. Both NIS master servers and NIS slave servers can respond to RPC requests, permitting multiple servers to be used both for redundancy and for load balancing. NIS is ''open,'' in the sense that Sun Microsystems has released information about the NIS API and its associated network protocols, and various free versions of Unix and Unix-like operating systems include source code to functional NIS implementations.

Like Hesiod, NIS suffers from some serious drawbacks when viewed as a complete SAR solution. While NIS is supported ''out of the box'' by a wider variety of operating systems and applications platforms than Hesiod, they suffer from similar security problems. NIS implementations used as the basis for SARs typically still expose secure authentication information on a possibly insecure network, and may in some cases promiscuously distribute passwd table information to systems outside a given NIS domain. Such exposure is widely considered to constitute a security breach, since publication of user authentication information (particularly encrypted passwords) can assist would-be intruders in the exercise of brute-force and cryptographic attacks against the authentication system.

NIS+: Security at a Cost

Partially to address concerns regarding the openness of the NIS and Hesiod security models, Sun Microsystems developed NIS+, a NIS (version 2) follow-on. NIS+ provides some of the same features as Hesiod and NIS, but does so through yet another completely different network distribution mechanism. Although similar to NIS in name and general functionality, NIS+ represents a major shift in Sun's approach to network-distributed passwd table information.

Like NIS, NIS+ relies on an RPC mechanism to achieve network distribution of arbitrary data, and like NIS, NIS+ is designed specifically to solve the problem of extending the traditional Unix passwd table

mechanism (and similar mechanisms) into a networked environment. NIS+, however, relies on a modified RPC mechanism called ‘‘Secure RPC.’’ In the traditional NIS model, arbitrary clients can bind themselves to any available NIS server and execute RPCs to access data housed on the server, leaving open the possibility of unauthorized accesses to secure authentication data. In the NIS+ model, clients and servers share a secret encryption key, allowing NIS+ clients and servers to cross-authenticate before information in any NIS+ tables is accessed. This additionally allows Secure RPC-based applications like NIS+ to take advantage of session-level encryption of network conversations, limiting or eliminating the exposure of secure information in plain text across a possibly insecure network. As a replacement for traditional NIS implementations in supported environments, NIS+ offers some significant security advantages.

NIS+ suffers from some serious drawbacks as the basis for a SAR in a heterogeneous network environment, however. To our knowledge, Sun Microsystems has not ‘‘opened up’’ the NIS+ protocols sufficiently to enable competing implementations. Thus, stable NIS+ clients are only available for use with Sun platforms (although note that there has been some limited success with reverse engineering NIS+ behavior for FreeBSD, Linux and other Free Unix platforms). While NIS+ servers *can* be operated in ‘‘NIS compatibility mode’’ to support non-Sun clients, running servers in compatibility mode erodes any security advantages offered by NIS+, making the NIS+-in-compatibility-mode server little more secure than a traditional NIS server.

Further, NIS+ has been demonstrated to suffer from serious performance degradation and stability problems in very large environments. The authors have particular experience with these problems, having been tasked with managing a NIS+-based passwd table distribution mechanism for a collection of approximately 200 Solaris machines for over two years. At Duke, NIS+ is in use as a mechanism for distributing passwd table information (but not, as we will discuss later, actual authentication information) for some 37,000 Solaris users, and we have observed a number of serious problems with this rather large NIS+ deployment.

While running performance of NIS+ in the presence of no underlying server or network problems has not been a significant issue, serious problems have arisen when one or more NIS+ servers have failed. Specifically, it has been our experience that multiple NIS+ servers cannot be expected to function reliably in failsafe modes. In theory, NIS+ shares the traditional NIS feature of supporting multiple servers within a given NIS+ domain, each of which can respond to service requests from clients in the event of a failure among the server group. In practice, our experience at Duke has been that NIS+ servers frequently exhibit failure modes in which, rather than refusing to respond to RPC requests and triggering client fall-back to other servers in the NIS+ domain, they respond *incorrectly* to RPC requests, resulting in clients receiving incorrect or invalid information. Further, because NIS+ (unlike NIS) offers no mechanism for forcing a particular client machine to direct its Secure RPC calls to a particular NIS+ server, we have found it to be exceedingly difficult to localize NIS+ failures when they occur, and nearly impossible to resolve them in an acceptable timeframe.

NIS+ has also exhibited serious administrative performance problems at times when it has been necessary to perform significant updates to large NIS+ tables. Transferring Duke’s 37,000-entry passwd table from the local NIS+ master server to its slaves across a 100-Mbit FDDI ring has been observed to take as long as 15 minutes. More significantly, regenerating NIS+ credential information (required by the Secure RPC framework underlying the NIS+ service) for all 37,000 system users has been seen to take in excess of 12 hours real time on an 85-MHz Sparc5 workstation, during which time no access is available to NIS+ objects within the directory tree rooted on the affected server. Solutions to these problems recommended by the vendor have essentially amounted to replacing NIS+ with another service. NIS+, we have been told, was designed to support tables containing up to 10,000 objects, and is known to suffer performance degradation when table sizes increase beyond that limit.

These issues have driven us at Duke to work toward eliminating as many dependencies on NIS+ as possible, and lead us to recommend against NIS+ as the basis for a SAR in any heterogeneous or large environment. NIS+ *can* be used effectively as a replacement for NIS or YP in small, homogeneous environments, but does not scale well to large applications and is not appropriate for use in heterogeneous computing environments.

RADIUS, etc.: SAR by Proxy

Hesiod, NIS, and NIS+ are all SAR-like services based on the concept of distributing a single database of authentication information (usually a Unix-style passwd table) to a variety of possibly heterogeneous systems. Authentication is still achieved, with these mechanisms, through client-based look-up of authentication information and direct comparison of authenticators presented by end-users against those recorded in central authentication databases. While this approach has some advantages, among them interoperability with a plethora of existing applications designed around the traditional Unix security model, other approaches to network-distributed authentication are available.

One such approach, authentication by proxy, is exemplified by the RADIUS (Remote Authentication Dial In User Service) authentication mechanism. In the RADIUS scenario, authentication is achieved by clients passing their users’ authenticators to a central

RADIUS server, which performs table look-ups to determine their validity, and returns an ‘‘accept’’ or ‘‘reject’’ response to the clients depending on the results of the RADIUS look-up. Client systems need not have direct access to any secure authentication tables, since actual authentication operations are performed by proxy on the RADIUS server. Rather than distributing the authentication database to client machines, RADIUS and related mechanisms pass individual users’ authenticator information to a central server for validation.

RADIUS has been implemented by a number of vendors of remote-access hardware (terminal servers, routers and other network devices) as a cost-effective mechanism for providing user authentication from within embedded applications.

Depending on the specifics of particular implementations, RADIUS can suffer from a variety of security flaws as a basis for an organizational SAR. In particular, although the network path between a RADIUS client and a RADIUS server can be and frequently is hardened, the connection between a RADIUS client and its user is often unencrypted. As such, widespread use of RADIUS has traditionally been confined to applications in which the connection between the authenticating agent and the RADIUS client can be expected to be invulnerable to passive monitoring attacks (e.g., dial-up terminal servers).

A mechanism similar to RADIUS has been in use at Duke University for a number of years in support of authenticating dial-up access to the institution’s campus-wide network. Users make dial-up connections to terminal servers on campus which, in turn, prompt for their authentication information and pass it to a Unix machine implementing a RADIUS-style authentication mechanism based on a secure third-party authentication service (Kerberos v4). Dial-up access to the terminal servers is either granted or refused based on the success or failure, respectively, of the Unix machine’s authentication operation. While this poses serious security issues in the Duke environment, traditionally it has been felt that the risk of authenticator compromise involved in passing authentication information in plain text over a presumably secure telephone network and across the hardened Ethernet network connecting the terminal servers with the authentication proxy is more acceptable than the risk of allowing unauthenticated dial-up access to the campus network.

Because they are widely supported in certain niche-applications (terminal servers, etc.), RADIUS and related authentication proxying mechanisms are likely to play a role in any large organization’s SAR. We do not, however, recommend that they be viewed as the basis for building a SAR, but rather recommend that they be used as an adjunct to other, more secure and more widely-applicable SAR solutions.

OTP: The Contrapositive of SAR

Yet another set of network-distributed authentication mechanisms which should be identified, perhaps less as SAR-building platforms than as competing solutions to many of the security problems addressed by properly-chosen SAR solutions, is the group of so-called OTP or One Time Password systems. Ranging in their implementation from totally software-based approaches (like S/Key) to totally hardware-based solutions (so-called ‘‘smart cards’’), OTP solutions address the problem of identity proliferation in a somewhat radical manner.

Rather than striving to reduce a plethora of system- and application-specific authenticators issued to each end-user to a single, highly-secure authenticator, OTP solutions strive to make authenticators secure by changing them each time they are used. In the typical OTP scheme, a given user will have as many passwords (although hopefully not as many login ids) as he or she has authenticated work sessions. Each time a user’s password (or, more generally, a user’s authentication information) is used, it is immediately invalidated, and new authenticators are issued for the user.

Provided that the mechanism by which new authenticators are issued to end-users is sufficiently unpredictable to third parties, such OTP schemes can eliminate the security risks involved in sending plaintext authentication information over an insecure network. By the time a nefarious observer becomes aware of the user’s identity and authentication information, it is no longer of any value.

Insofar as most OTP systems rely on some central authority to manage and coordinate the issuance and revocation of single-use authenticators, they may realistically be viewed as providing a sort of SAR, and to the extent that OTP systems maintain consistency of user identities (login ids, for example) through time, they can form the basis of something very similar to a SAR.

Software-based OTP mechanisms, such as S/Key, typically work by pre-assigning a sequence of authenticators to each individual, a list of ‘‘upcoming’’ passwords, as it were. Each user must periodically query the S/Key service for a new list of future authenticators, and must refer to the list whenever performing authentication operations on supported systems. Similarly, all supported systems must be accessible at an administrative level to the S/Key service, so that authenticators may be invalidated and replaced as they are used.

Obviously, such list-based OTP solutions can only be as secure as the mechanisms by which these lists of authenticators are transported. Requesting a new authenticator list over an unsecured network channel, printing an authenticator list on an insecure printer, or storing an unencrypted list of future authenticators online or in some other insecure fashion undermines the security of the entire list. So long as

sufficiently secure channels are used to manipulate authenticator lists, however, list-based OTP mechanisms can provide an inexpensive enhancement to the overall electronic security of an organization.

By contrast, hardware-based OTP solutions (like Security Dynamics' SecurID product) typically generate new authenticators as they are required, thus eliminating the need for persistent lists of single-use authenticators. In these schemes, each user is issued an electronic device (typically a roughly credit-card-sized microcomputer) equipped with enough intelligence to calculate a complex and unit-specific cipher which is in turn used as the owner's authenticator. Depending on the specific implementation, the cipher may be a function of the current time (in which case the user's ''smart card'' must have a means for synchronizing its clock with the central security system), a function of some random challenge string presented to the user at the start of an authentication operation (in which case the user's ''smart card'' must offer some means for ascertaining the value of the challenge string), or a function of both. Additionally, smart card systems may require users to provide the interface between cards and authenticating systems (by manually responding to OTP challenges presented by target systems) or may make use of additional hardware to directly connect authenticating systems to smart cards.

Provided that the cipher calculated by the user's smart card is sufficiently secure (that is, provided that it is both cryptographically secure and sufficiently difficult to reproduce using other hardware) such systems can offer the same sorts of advantages list-based OTP solutions like S/Key offer. In addition, hardware-based systems offer the advantage of requiring little or no change in the behavior of end-users; users need not learn to operate a new software system in order to manage their authentication information. Rather, they may simply substitute a password provided on demand by their personal authentication device for a traditional remembered password.

Hardware-based OTP solutions typically involve much higher installation and deployment costs than software-based solutions. Especially in large organizations, the cost of acquiring and issuing thousands of smart cards and possibly card reading interfaces may be prohibitive, and in addition, the long-term cost of maintaining and replacing malfunctioning or stolen authentication devices can be exorbitant. Similarly, although software-based OTP solutions may be deployed without significant capital outlay, even within large organizations, the cost associated with retraining users and supporting their use of a more complicated authentication scheme may be prohibitive.

Unlike distributed database SAR solutions and proxy solutions such as RADIUS, OTP solutions rely on ''something the user has,'' rather than ''something the user knows'' to achieve strong user authentication. Depending on the extent to which users protect their OTP lists or smart cards, such ''bearer bond'' authentication strategies can be either more or less secure, overall, than equivalent password-based systems. Increasingly, OTP mechanisms are being viewed as an adjunct to, rather than a replacement for more traditional password systems. While the cost of deploying an OTP solution ubiquitously across a large organization may be prohibitive, the cost of deploying such a solution for a few, key users within the organization (presumably those whose level of authorization makes the strength of their authentication particularly important) may be less so. In many instances, hardware-based OTP solutions are being deployed *in addition to* traditional password-based mechanisms in order to achieve an even higher level of certainty in authentication than either mechanism can provide alone.

PKI: SAR Meets Star Wars

Yet another, arguably much more secure approach to network-distributed authentication involves the use of digital certificates under the auspices of a public key infrastructure, or PKI. Although it has yet to achieve the status of a true ''standard,'' the most widely-accepted approach to digital certificate/PKI authentication is the proposed X.509 standard.

In this scenario, authentication is achieved by presenting a digital certificate to the challenging system or application. This digital certificate is actually a structured block of data identifying the certificate's owner and typically including the owner's public key which has been digitally signed (using one of a number of related public-key encryption technologies) by the certificate's issuer, the user's certificate authority or CA. Presented with this digital certificate, a cooperating system can verify (by decrypting the certificate with the CA's public key) that the certificate is genuine (i.e., that it was issued by the certifying authority). The CA can frequently be queried to verify that a particular certificate is valid (i.e., that it has not been revoked). Certificate authorities, in this scenario, must each have their own public key/private key pairs (to effect signing of the certificates they issue), and must act as key distribution agents, providing a central repository for the retrieval of public key information. Participating systems and applications then ''trust'' particular certificate authorities, accepting valid certificates issued by those CAs, usually on the basis of the CA's being known to use trusted methods to identify individuals before issuing them certificates.

Public key certificates offer a number of advantages as an authentication mechanism. Being based on public key encryption mechanisms, certificates are highly cryptographically secure. Forging a public key certificate without prior knowledge of both a user's private key and his CA's private key is virtually impossible. Further, certificates offer some of the advantages of hardware-based OTP systems, in that authentication is achieved based on a user's

possession of something (his or her certificate) rather than his knowledge of something (his or her password).

Additionally, digital certificates provide a natural means for engaging in secure communications over a possibly insecure network. Once a user's public key certificate has been exchanged and validated as proof of authentication, a shared encryption mechanism is available to both the authenticating user and the system to which he or she is authenticating. The user may encrypt data in his or her private key (as distributed by the user's CA) to ensure the authenticity of transmissions (data encrypted in the user's private key can only be decrypted using that user's public key). Participating systems may encrypt data in the user's public key, ensuring that their transmissions can only be decrypted using the user's private key.

Public key certificate systems are not, however, a perfect solution to the SAR problem. In order to deploy a SAR based on public key certificates, an organization must first develop and deploy a rather complicated set of support services, a public key management infrastructure, which may include, in addition to one or more local Certificate Authorities, a key escrow system (for the secure storage and retrieval of private key information) and mechanisms for updating, invalidating, and re-publishing public keys. Security of the various portions of the public key infrastructure becomes critical, since compromise or impersonation of a part of the public key infrastructure can directly undermine the security of any authentication mechanisms designed around it.

In certain environments, it may be feasible to rely on an external CA, effectively outsourcing the work involved in setting up and maintaining a public key infrastructure. In many organizations, however, the need for guaranteed access to the certificate authority or special requirements for user identity verification and certificate maintenance (frequent certificate revocation, for example) may make an off-site CA untenable. On the downside, reliance on an organization-specific CA may cause difficulties for users who interact with systems which do not trust the organizational CA, resulting in individuals having to juggle multiple personal certificates (one for use within the organization, for example, and one for use outside the organization) and undermining the intent of a SAR.

Moreover, few applications and no common operating systems are currently built to support PKI certificates as the basis for user authentication. So-called ''personal certificates'' are supported by a wide range of web-based applications via the intrinsic support for certificates provided through common web browsers (Netscape, Internet Explorer), but non-web-based applications and systems typically offer no mechanism for supporting certificates as an authentication tool. A SAR based entirely on the exchange of public key certificates in any but the most homogeneous of organizational environments would require significant re-development of common applications and systems.

PKI certificates further pose problems in environments where users are highly mobile. Because of their complexity and size, certificates cannot be feasibly memorized or re-entered by their owners, and so must be stored electronically. In non-mobile user environments, certificates can reasonably be stored on users' preferred client machines, where they can be reliably accessed at all times. In more mobile environments, where a single user may work from any of a number of locations, access to a user's PKI certificate can become more complicated. Hardware-based solutions, in which certificate information is stored in ''token cards'' carried by users, can make certificates easily transportable, but are expensive to implement across large organizations. Software-based solutions involving key and certificate escrow systems can make certificates network-accessible, but require some form of alternative authentication in order to ensure secure access to escrowed key/certificate information. These difficulties are specific, of course, to end-user authentication systems; public key certificates used to authenticate systems to one another need not be transported from one system to another and so do not suffer from these complications.

In spite of these difficulties, it is clear that public key certificates will play an increasingly important role, particularly in the realm of secure electronic commerce applications. Any SAR or SSO infrastructure developed today will need the ability to use public key certificates where they are appropriate. However, building a SAR based solely on public key certificates and a central CA will only be feasible for the short term within certain organizational environments. The authors believe that the cost of deploying, managing and maintaining a public key infrastructure currently makes certificate-based authentication mechanisms unattractive from the point of view of the typical system administrator.

Kerberos: Established Technology, Secure Authentication

Yet another secure network-based authentication mechanism, and one which the authors recommend as the preferred basis for developing SARs within most organizations, is Kerberos. Developed at MIT under the same Athena project which spawned the Hesiod system discussed earlier, Kerberos has been in use within a large number of organizations for a number of years. The Kerberos technology, represented by MIT Kerberos versions 4 and 5 as well as by the security service associated with the Open Group's DCE infrastructure, is both widely-understood and reasonably impervious to common methods of attack.

The impetus for the development of the Kerberos authentication model was a simple question: how can

users be authenticated to systems across insecure networks without exposing their authentication information to would-be attackers monitoring network traffic? Relying in part on previous work by Needham and Schroeder, developers at MIT designed the Kerberos model as a means of authenticating clients without resort to passing re-usable authentication information (passwords, etc.) over an insecure network.

In the Kerberos approach, clients and servers are loosely grouped together into ''realms,'' with each realm containing at least one shared security server. This security server, called a KDC or Key Distribution Center, shares a secret encryption key with each authenticatable user and with each participating network service provider within the realm. These keys are used as shared secrets to effect user authentication and to issue reusable authentication credentials called ''tickets.'' Users and service providers within a given realm trust the realm's KDC, and in some cases, may trust foreign KDCs through a chain of trust relationships and shared keys between KDCs in multiple realms.

Kerberos achieves authentication through the use of shared secret keys. The model is based on the simple realization that if two parties wish to verify one another's identities, they may do so securely by exchanging information strongly encrypted in a shared secret key. The party initiating the authentication process can send a message encrypted in the shared secret key, and if the responding party is able to properly decrypt the message, both parties can be reassured of one another's identities. Provided that the key is actually a secret shared solely between the two parties, the ability to decrypt one another's messages is sufficient to prove authentication. If part of the encrypted information passed in the original exchange is further encrypted in a secret key known only to the originating party, the originating party can subsequently verify the origin of the initial message.

In reality, the Kerberos model is made more complicated by the need for more than two parties to authenticate to one another securely in real environments (necessitating the use of persistent ''tickets'' as records of prior authentication), and by the need to address certain security vulnerabilities intrinsic to its use in insecure network environments (network replay attacks, dictionary-based key-guessing attacks, etc.).

Conceptually, the KDC can be viewed as providing two different but related services: an initial ''authentication service'' or AS, used to perform initial authentication of users and issue tickets for the ticket granting service, and a ticket granting service or TGS, used to issue tickets for other participating services. In most existing implementations of the Kerberos model, the AS and TGS reside on the same secure host(s), and in most cases the two are implemented using the same executable code.

In the classical Kerberos authentication model, initial authentication involves the acquisition of a ''ticket granting ticket'' or TGT (basically, a ticket for the ticket granting service) from the KDC's AS. This is accomplished in two steps. In the first step, the client sends an authentication request to the AS, indicating the user for whom a ticket granting ticket is sought and providing information (time stamps, etc.) useful for validating the request. The AS responds with a TGT which is encrypted in the user's secret key. The client's ability to decrypt this TGT is the basis for the client's proof of authentication, since only the user and the KDC are presumed to know the user's secret key. Encryption is usually achieved using the DES encryption algorithm, but can in theory be achieved using any symmetric-key encryption scheme.

Enclosed in the ticket granting ticket are a number of pieces of information, of which five are particularly important: an actual user credential identifying the authenticated user (encrypted, itself, in the server's own secret key, to prevent foreign construction of credentials), a timestamp issued by the KDC identifying the time at which the ticket granting ticket was issued, a lifetime value indicating how long the ticket is to remain valid, a checksum useful for ensuring that the ticket has not been modified from its original content, and a randomly assigned ''session key'' for use as a shared secret between the now-authenticated user and the KDC. If the client is able to decrypt the ticket granting ticket correctly (i.e., if the resulting ticket contains a valid checksum) and if the timestamp on the ticket matches (within a short window) the current local time on the client, the ticket is accepted as valid. If any but the correct key is used to decrypt the ticket, or if the data within the ticket is changed from what was originally issued by the KDC, the resulting decrypted key will not match its checksum and the ticket can be identified as invalid. Together, encryption in the user's secret key and the presence of the checksum ensure the integrity and confidentiality of the TGT.

At no time during the initial ticketing transaction does the user's secret key information pass over the network, and at no time do reusable authentication credentials pass over the network unencrypted. As such, the initial ticket exchange can in principle be performed securely over an insecure network.

Once acquired, the user's ticket granting ticket can subsequently be used in additional ticket exchanges with the KDC, usually to acquire so-called ''service tickets'' as proof of authentication for use with services other than the TGS. Subsequent ticket exchanges proceed similarly to the initial ticket exchange, with requests being made to the TGS rather than the AS, and with requests specifying a target service in addition to a target user. Authentication for service ticket requests may be performed in the same way as authentication for initial ticket requests is performed (using the client's knowledge of the user's

secret key to authenticate the user), but more commonly, a previously-acquired TGT is used as proof of authentication. The TGS responds to a service ticket request with a service ticket containing information similar to that in the TGT. The response is encrypted in the session key shared between the KDC and the authenticated user, and contains within it information encrypted in the secret key shared between the target service and the KDC. This service ticket can then be used as part of an authentication transaction with the target service, which can ensure that the ticket being presented to it is valid based on its being encrypted in the target server's secret key (a key known only to the target server and the KDC).

Three different implementations of the Kerberos model are currently in common use: the ''original'' Kerberos version 4, Kerberos version 5, and the Open Group's DCE security server. All are built atop the general framework outlined above, although each differs from the others in the specifics of the communication protocol used and the details of the contents of tickets. Kerberos version 4 is perhaps the most commonly-deployed implementation of the model, having been generally available since the mid-1980s. As Kerberos V4 came to be deployed on larger scales, it became apparent that the original implementation suffered from some serious security flaws, including a particular susceptibility to dictionary-based attacks. Kerberos V5 was developed in the early 1990s to address these concerns, and to provide some additional features (e.g., better cross-realm authentication, forwardable tickets, and renewable tickets) required by new applications of the model. The addition of support for prior encryption of ticket granting requests and changes in the underlying ticket exchange protocol make Kerberos V5 less vulnerable to certain common dictionary-based cryptographic attacks. The DCE security service was developed as part of the Open Group's Distributed Computing Environment, a larger (and some have said, too large) project striving to provide an infrastructure for secure, cross-platform distributed computing. Loosely based on an intermediate version of Kerberos V5 from MIT, the DCE security service is conceptually similar to Kerberos V5, but relies on the DCE ''secure RPC'' mechanism as the underlying conduit for ticket exchanges.

Kerberos offers a number of positive features for both system administrators and end-users as the basis for an organizational SAR. The Kerberos model provides the advantages of strong authentication based on strong encryption without the overhead and complications exhibited by current PKI implementations. In principle, only one dedicated network server must be installed, secured, and maintained to support a Kerberos infrastructure, although in practice multiple ''clone'' security servers are usually deployed to provide redundancy and availability. Like PKI/certificate based authentication mechanisms, Kerberos provides a natural mechanism for ensuring the privacy and integrity of application-level data exchanges over an insecure network (via the session keys distributed with Kerberos tickets), and provides a mechanism for bipartisan authentication (i.e., the model supports both the authentication of client users to server systems and the authentication of server systems to client users).

Kerberos boasts an enormous installed user base, and is one of the most proven network-based authentication schemes available. From the standpoint of the systems administrator, who may be held ultimately responsible for both the security and availability of systems participating in the SAR, this historical track record can be a significant advantage. Kerberos is further supported by well-established Internet standards, and as such forms the basis for continuing development efforts world-wide. New implementations of the Kerberos model continue to be developed (viz., work at KTH in Sweden and recent work toward both Tcl and Java-based implementations of Kerberos V5). Kerberos is already supported as an authentication mechanism by a number of high-profile application vendors (Oracle, SAP), a variety of common open application servers (IMAP, POP, ACAP) and is supported natively on a number of operating system platforms (AIX, Solaris). Upcoming versions of some very popular operating systems (Windows NT, Novell Netware) are also expected to include native support for Kerberos as an optional authentication method.

Kerberos does suffer from some well-known deficiencies, both as an authentication system and as the basis for an organizational SAR. Kerberos is, at its base, a password-based system, and as many have pointed out, it is subject to a variety of password-guessing attacks. Further, the model (particularly in its implementation under Kerberos V4, but to some extent still in Kerberos V5 and the DCE implementation) is subject to certain types of replay attack; a determined attacker may, under certain circumstances, be able to circumvent the replay protections built into the Kerberos protocol and for a short time masquerade as an authenticated user by replaying part of a previous ticket exchange on an insecure network. As Bellovin and Merritt, among others, have pointed out, Kerberos is also subject to environmental attacks. Depending, as it does, on time synchronization between participating clients and servers, Kerberos security can be undermined in the presence of insecure timekeeping mechanisms. Additionally, since Kerberos session keys may be used to encrypt multiple messages between a single client and server during the lifetime of a given Kerberos ticket, session keys may be vulnerable to cryptographic attack by sufficiently determined enemies.

Like PKI solutions, the security of a Kerberos-based SAR is wholly dependent on the security (both logical and physical) of the systems making up the authentication infrastructure. The security of the KDC within a given realm is as important in the overall security of a Kerberos-based SAR as the security of

any portion (CA, key escrow systems) of a shared PK infrastructure. The authors believe that, because of its relative simplicity and wide availability, a Kerberos SAR can be more easily (and thus, more probably) secured against infrastructural attack than a more complex, less widely-implemented certificate-based SAR. As certificate-based systems come of age, however, systems administrators may find that PKI-based SAR solutions gain ground in this respect.

Like the PKI/certificate approach, the Kerberos approach to building a SAR is not without development costs. Applications must be designed or modified to support Kerberos (must be ''Kerberized'') in order to participate in a Kerberos-based SAR, and although a number of applications and operating systems already provide some measure of Kerberos support, many do not. Depending on an organization's installed base and usage patterns, adoption of a Kerberos-based SAR may require significantly more or significantly less retrofitting of existing applications and systems than a PKI-based SAR.

Experience at Duke, where Kerberos has been in use for a number of years in a SAR supporting a variety of public computing labs on campus, has shown that Kerberizing applications whose intrinsic authentication models are relatively modular, while not trivial, is relatively straightforward. The published Kerberos API, with its support for RFC 1510's GSS-API standard, makes Kerberizing most applications a matter of adding on the order of ten or twenty lines of C to server and client code. Kerberos extensions developed at Duke (including the Exu system co-developed by one of the authors) have simplified the integration of Kerberos authentication into a variety of systems-related tasks. It should be noted, of course, that not every application or system can be modified at every site. In the case of proprietary software and systems, the system administrator must frequently rely on cooperation from one or more vendors in order to implement support for a Kerberos- or PKI-based SAR. At Duke, the authors have been fortunate in having access to source code for some proprietary systems, and in having the support of upper management in the use of free and open software wherever possible.

Likewise, experience at Duke has shown the Kerberos model to be extremely scalable, supporting very large numbers of authenticatable entities (termed ''principals'') with little or no measurable degradation in performance or stability. Currently, the ''ACPUB.DUKE.EDU'' Kerberos realm supports in excess of 37,000 users and provides secure, authenticated access to a range of applications from electronic mail to dial-up networking across a variety of computing platforms, from Macintoshes to Unix-based file-servers. This is not to say that there are not costs which increase as the user-base for a SAR grows (certainly, end-user support costs can increase dramatically) but rather that the administration of a Kerberos infrastructure supporting tens of thousands of users is not significantly more complex or time-consuming than the administration of a similar infrastructure supporting only hundreds of users.

Many of the security vulnerabilities in Kerberos are addressed to varying extent in recent implementations of the authentication model; Kerberos V5 is significantly stronger as an authentication mechanism than Kerberos V4, and the DCE implementation of Kerberos offers even greater protections against certain forms of attack by modifying the ticket exchange protocol in some significant (if incompatible) ways. Kerberos implementations continue to evolve, with MIT and others investigating extensions to the Kerberos protocol to support more cryptographically secure authentication mechanisms and to integrate support for newer authentication approaches (including digital certificates and PKI-based authentication).

The authors believe that, for most organizations, a Kerberos-based SAR can provide more than adequate security with greater confidence and less administrative overhead than other SAR mechanisms. While PKI-based authentication mechanisms may offer advantages in the realm of electronic commerce, where users may not be known by the organizations they interact with until the need for authentication arises, they do not yet offer the proven reliability nor the existing installed base of standard implementations Kerberos boasts.

SSO Alternatives: The Dream of a Single Sign-On

Many of the SAR solutions discussed above might reasonably be viewed as also providing the benefits of a full SSO solution. In the Kerberos model, for example, a single authentication to the SAR acquires the client reusable credentials sufficient to authenticate to any Kerberized application without re-entering authenticator information. Likewise, in the PKI model, possession of a personal PK certificate may allow a user to freely authenticate to a variety of supported applications without re-entering passwords or other authentication information.

Nevertheless, in all but the most restricted of environments, it is unrealistic to expect that a SAR can provide a complete SSO solution for an organization. In the face of a heterogeneous computing environment comprising many different systems and applications, it is unlikely that a SAR can support the needs of the entire organization. Further, while a SAR may provide a basic level of SSO functionality (one authentication operation per user per work-session), it cannot usually provide the look-and-feel features demanded by many users. Systems administrators may find that providing a mechanism whereby users need only authenticate themselves once per work session is not enough to satisfy their user populations; users may demand not only strict SSO functionality but also a simplified authentication interface common to multiple environments.

Hence, system administrators charged with designing and building SSO infrastructures must frequently look beyond SAR options and investigate true SSO approaches. Common SSO approaches fall into three main categories: those which rely on an underlying SAR to facilitate authentication into multiple services at login-time, and those which rely on some centralized authenticator repository (a ''key box'') to provide multiple applications and systems with the ''illusion'' of a shared, single sign on, and hybrid solutions incorporating features of both the SAR-based and key box-based solutions.

Entry-Point Authentication: Pay No Attention to the Man Behind the Curtain

One common approach to providing SSO functionality involves the modification of common entry-point applications (login programs, etc.) to perform multiple authentication operations each time a user initiates a work session. In this approach, a number of different systems and applications sharing user information through some sort of SAR pass authentication information between one another in order to effect the appearance of a single sign-on solution.

Typical examples of this methodology include the Windows NT ''layered GINA replacement'' mechanism and the Kerberized Unix login mechanism.

The Windows NT user authentication mechanism (GINA) is designed in such a way that multiple different authentication modules can be invoked in succession when a user logs into an NT machine. Provided that a single common login and password are sufficient to authenticate a user to, for example, both an NT domain and a Novell NDS tree, layered GINA modules can be used to provide a sort of single sign-on appearance to the end-user.

Similarly, the MIT Kerberos distribution includes standard replacements for the normal Unix ''login'' program which, rather than checking user's identities against a local or distributed Unix passwd table, perform Kerberos authentication using the login and password supplied by the user. In essence, the Kerberized Unix login replacement is a sort of SSO mechanism, presenting an SSO interface for authenticating to both a Kerberos realm and an individual Unix machine.

Certainly, these methods have a place in organizational SSO planning. In order to implement a Kerberos-based SAR, for example, in a fashion palatable to general users, Kerberized entry-point applications (login programs, etc.) must be made available across as many platforms as possible. These sorts of approaches offer little beyond what can be achieved through a SAR alone, however, and do nothing to achieve a unified look-and-feel across applications or systems.

More importantly, these solutions still rely on underlying application- and system-level support for some form of SAR, and in all but the least complicated situations, require significant re-engineering of entry-point code on multiple systems. Clearly, while the entry-point approach to single sign-on is of interest, it is not a comprehensive solution to the demand for SSO functionality.

The Key Box: SSO in a Fire Safe

Another common approach to the SSO problem, and one which seems to have been *en vogue* among vendors in recent years, is the ''key box'' mechanism. A central authenticator repository is established containing each user's various application- and system-specific authenticators (login/password pairs, certificates, etc.). Rather than the user authenticating to individual applications and systems directly, the user authenticates once to the central SSO server, which in turn sends the necessary authenticator information to his or her client machine to allow an SSO client application to authenticate to other systems and applications on the user's behalf. Typically, the SSO client application populates the user's graphical desktop with icons (or otherwise makes available a list of executable links) for each of the systems the user has SSO access to. Launching one of the ''virtual'' applications set up by the SSO client causes the appropriate application to be started *and* automatically authenticates the user to the application using whatever authenticator information was provided by the SSO server. In effect, the SSO client provides the user with the appearance of a single sign-on by performing authentication operations on the user's behalf.

Unlike the entry-point authentication approach, the key box approach doesn't require the prior existence of a SAR. Because the SSO server can store multiple different authenticators for a single user (one for each different system and application the user is authorized to access), there is no need for a SAR as such. Multiple authentication operations are performed during each work session, but they may be performed entirely without the user's knowledge.

While key box approaches to the SSO problem can provide a high level of functionality for end users, they can also pose some difficult challenges for system administrators, both in the realm of providing adequate security and in the realm of managing users' authentication information.

Clearly, the security the key box or SSO server system is of primary importance, since compromise of that system would lead directly to compromise of all systems and applications participating in the SSO solution. Further, ensuring the security of communications between the SSO server and its clients is of paramount importance. Depending on the specific implementation, users' authenticators may be repeatedly exposed to passive network attacks as they are transferred from the SSO server system to various client machines. Since the SSO client must typically store reusable authenticators for its users locally in

order to provide the appearance of a single sign-on environment, SSO clients must themselves be secure, lest client-based attacks permit the compromise of all of a particular user's authenticators.

Additionally, SSO clients must be specifically designed to interoperate with target applications and systems. In order for an SSO client to perform authentication on behalf of the user, it not only must have access to the user's authenticators via the SSO server, but also must have special knowledge of the native authentication interfaces used by each SSO-supported application. As applications and systems change, SSO clients may need to be modified or replaced in order to support new authentication interfaces.

Likewise, key box-based SSO solutions introduce special problems for user and authenticator management. While the key box-based SSO approach can simplify authentication for the end user, it does not by itself address the system administrator's need to reduce the number of individual authenticators which must be validated, issued, and maintained for each user. Further, the SSO software must provide some mechanism for maintaining (adding, removing, and updating) authenticators stored in the central SSO repository, and should ideally provide some mechanism for ensuring that the authenticators housed in the central SSO repository are kept synchronized with application- and system-specific authentication information. In some implementations, end-users may not be privy to their own authenticator information; login ids and passwords, for example, may be chosen and maintained by the SSO server. While this can simplify the management of a large set of users or participating systems, it can lead to serious problems if user's clients are not carefully managed. If a user's client loses the ability to interact properly with the SSO server and becomes unable to retrieve authenticators from the server, the user has no means of authenticating to other systems. In many cases, the benefits of central user management afforded by such SSO systems can be completely eroded by the associated increase in system management effort needed to ensure proper configuration of client systems.

GSO: IBM's Lock Box Solution

IBM's Global Sign-On (GSO) product is a typical implementation of the key box approach to SSO. In the GSO implementation, users' authentication information is warehoused on one or more central GSO servers, which also provide DCE-based authentication services. Users install GSO clients on their workstations which: manage the initial authentication operations necessary to access the GSO server, present a virtual desktop prepopulated with icons for all the applications for which the GSO server holds the user's authenticators, and transparently perform authentication on behalf of the user for each supported application when it is invoked. Once a user has started the GSO client and authenticated to the GSO server, the user need not perform any further authentication operations to access supported applications. Further, the GSO server provides built-in mechanisms for automatically responding to password aging as well as a user interface for changing passwords on supported systems. Recently, IBM has begun marketing the GSO product in conjunction with Tivoli remote system management products in an effort to provide a comprehensive user management/SSO solution.

The IBM GSO product is exemplary among its competitors in its attention to security and its integration with a remote management facility (Tivoli). By relying on DCE for its primary user authentication, the GSO product can provide some of the advantages of a strong SAR in addition to providing the advantages of a targeted SSO solution. One of the strengths of DCE (and its underlying Kerberos structure) as a method of primary authentication is the natural way it provides for session-level encryption of data flowing over a network. IBM's GSO takes advantage of this strength to provide a fair level of security in the transmission of sensitive data (logins, passwords, etc.) between the GSO server and its clients.

By integrating the product with Tivoli, IBM circumvents two of the most difficult administrative problems associated with the use of targeted SSO solutions: the distribution and maintenance of SSO client software across an organization and the creation and management of system- and application-specific authentication identities across distributed systems. The inner workings of Tivoli's TME-10 product are beyond the scope of this paper, but in brief, Tivoli brings to the GSO product a mechanism for centralized installation and upgrade of GSO client software across a large organization, and provides a mechanism for creating and registering with the GSO server new user identities across a range of systems.

Similar solutions (differing mostly in the methods used to authenticate users to the central key box server and the mechanisms by which application-specific authentication operations are proxied by the SSO client) are offered by other vendors, including Platinum Technologies (in a product called Platinum AutoSecure) and CoreChange (a vendor specializing in SSO solutions for the health-care industry).

Hybrid Solutions: Best of Both Worlds

SSO solutions share a primary goal: simplifying authentication operations for the end-user. Issues of actual security and manageability are frequently of secondary importance in the design of packaged SSO solutions. At times, the system administrator is confronted with an apparent trade-off between addressing the demands of end users through an SSO solution, on the one hand, and managing and securing systems adequately through a SAR, on the other.

The authors believe that a hybrid approach to the SSO problem is often the system administrator's best

option. Such a hybrid approach would rely on a strongly-secure SAR solution to provide the appearance of a single sign-on for the most critical systems and applications within an organization and would also rely on a possibly-integrated SSO solution to support those systems which cannot feasibly be integrated into a SAR and to provide end-users with the look-and-feel features they want. By building a key box SSO solution on top of a secure SAR mechanism (such as Kerberos), the system administrator can often address organizational security and management requirements as well as end user ease-of-operation requirements with a single comprehensive solution.

SnareWorks: Hybrid Key Box and SAR

One such product, which the authors have recommended for use at Duke, is the SnareWorks product available from IntelliSoft, Inc. The SnareWorks solution sits atop a minimal DCE infrastructure (to provide security and authentication services), and provides both end-to-end encryption of network traffic and authentication/single sign-on services to supported applications.

In the SnareWorks approach, network servers are equipped with a SnareWorks server application which takes control of incoming network connections on the server. Likewise, network clients are equipped with a ‘‘thin’’ SnareWorks client application which takes control of outgoing network connections on the client. A centrally-managed SnareWorks ‘‘master server’’ manages information about the security, authentication, and single sign-on requirements of individual servers and clients, and provides for rather fine-grained definition of security rules. A particular IP port on a particular server can be configured, for example, to require authentication before accepting incoming connections, and to perform different levels of encryption at different times of day, when talking to different client machines, and when manipulating connections authenticated as different users.

When a ‘‘snared’’ client machine connects to a ‘‘snared’’ server, the SnareWorks software on the two participating machines negotiates the levels of security to be provided for the new connection: whether to perform network-level encryption, and if so, using what keys and what encryption mechanisms, whether to require authentication before establishing the connection, and what, if any, ‘‘single sign-on’’ services to provide over the connection. If the machines agree that authentication is required, the SnareWorks client checks to determine whether authentication credentials have already been obtained from the associated DCE cell, and if they have not, automatically prompts its user for authenticators with which to acquire credentials. If credentials are already cached within the SnareWorks client, the client engages in a normal DCE/Kerberos authentication interchange with the server to prove its identity. Depending on the configuration of the SnareWorks client and server in question, unauthenticated connections may be refused, or may be treated differently from authenticated connections.

SnareWorks provides single sign-on features through a secure key box mechanism based on the necessarily-present DCE registry. Using extended registry attributes, the SnareWorks software stores authenticator information for various users on various systems and applications in the DCE cell’s central registry. On snared servers for which single sign-on support is available, small loadable modules (termed ‘‘Program Support Modules’’ or PSMs) are installed which encapsulate the information necessary for the SnareWorks server to perform application- or system-level authentication on behalf of pre-authenticated users. Single sign-on is effected through the PSM when the appropriate SnareWorks server retrieves (via a DCE Secure RPC channel) authenticator information for an already-authenticated user and intercepts the authentication transaction on the server. In essence, the PSM running on the server plays the role of the client, providing a single sign-on appearance without requiring any interaction on the part of the client. SnareWorks clients provide a straightforward user interface for creating and modifying user-specific single sign-on information in an associated DCE registry, and SnareWorks PSMs provide support for server-driven authenticator management (single sign-on information updates resulting from authenticator or password expiration, for example). SnareWorks comes pre-packaged with PSMs for a number of common network-based applications and protocols (telnet, rlogin, ftp, LDAP, X11, Oracle-8, etc.) and offers a relatively simple API for constructing PSMs to support single sign-on features for more exotic or site-specific applications.

With the addition of a SnareWorks Web server interface, the SnareWorks software can be used to provide secure authentication for and highly-configurable control over web-based CGI applications running on standard HTTP servers. Unlike other SnareWorks products, the SnareWorks Web server can provide these features to end-users running on clients which do not themselves have SnareWorks client software installed – the SnareWorks Web interface only requires the existence of a web browser on client machines. With the addition of an optional SnareWorks CA, the software can be made to interoperate with certificate-based authentication and authorization strategies and can be made a part of an organizational PKI.

The SnareWorks approach offers a number of advantages both as a network security mechanism and as a single sign-on solution. Based on a secure strong authentication mechanism (DCE), SnareWorks can provide all of the features of a secure SAR for applications which have native support for Kerberos or DCE authentication. Further, SnareWorks provides a flexible, configurable mechanism for enforcing encryption of application data flowing over a possibly-insecure

network. SnareWorks also provides a scalable and retargetable mechanism for providing a single sign-on presentation to end-users, all without requiring modification of any application client or application server code. Security management is centralized through a simplified administrative user interface, allowing security rules to be changed in one location and propagated to snared servers throughout an organization. Likewise, management of the authentication information stored on an associated DCE cell is simplified through the SnareWorks administration interface which can, if so desired, be configured to provide for automatic registration of users within a DCE cell.

The approach does have some shortcomings, however. Currently, IntelliSoft offers client support for Solaris, AIX, and HP/UX workstations and Microsoft Windows 95 and Windows NT client machines. Solaris, AIX, HP/UX, and Windows NT servers are also supported. Other platforms (including Apple's MacIntosh) are not supported natively, although certain features can be presented to unsupported clients through the SnareWorks web interface. Additionally, although source code to all of IntelliSoft's PSMs is available, the solution is a proprietary vended application; sites intent on having source code available locally for all critical software may find it difficult to justify this approach.

The Duke Authentication Project: A Short Case Study

As mentioned above, Duke University is currently involved in a major project to provide institution-wide authentication, security, and single sign-on services. The project was originally conceived by senior management within the University's Office of Information Technology as a means of simplifying use of certain newly-deployed enterprise-wide applications (e.g., SAP/R3, PeopleSoft, Lotus Notes). Initially, the project was directed at finding an SSO-only solution (with or without implementing a SAR) which would allow users to authenticate only once per work session, regardless of the applications they might be using.

A review committee was formed, with the authors as co-chairs and comprising representatives from major computing organizations on campus involved in enterprise-wide applications deployment, and tasked with investigating available technologies and returning an implementation proposal. Originally, this process was expected to take only two or three months, and result in a plan which could be implemented in roughly the same timeframe. After some initial discussions within the committee, it became clear that the process would be more involved than had previously been expected.

As the committee process unfolded, it became clear that different constituencies within the organization had different priorities and different requirements for an ''acceptable'' solution. Those of us involved in system administration were primarily concerned with ensuring the security of the authentication service and those applications and systems it supports. Those involved in application programming were concerned with reducing the impact of any new infrastructure on existing applications and systems. End-user support staff were primarily concerned with ease of use issues, while management was primarily concerned with finding a cost-effective solution.

Starting from these different points of view, the committee were able to arrive at a number of site-specific requirements for an acceptable solution. Among these were:

- The absence of a ''flag day'' visible to end-users. Whatever solutions might be deployed, they should be deployed with a minimum of disruption to existing end-user work-flow patterns, and should be introduced into the existing environment in a stepwise fashion, allowing users to adjust gracefully to changing usage patterns.
- Compatibility with existing infrastructures. At the start of the investigation, Duke already had a large investment in Kerberos technology, having operated for a number of years a central Unix computing facility using Kerberos version 4 to support in excess of 35,000 users. Further, a number of applications were already in use across the enterprise, and would need to be supported by any successful SSO solution.
- The absence of any recertification of users. The University had, at the inception of this investigation, already performed significant identity verification for well-over 35,000 of its constituents in order to issue them Kerberos authenticators. The cost of this verification (which had originally been performed over a period of many years) was significant, and the committee as a whole agreed that any solution which would require the recertification of existing users (i.e., which would require issuing new authenticators to individuals already in possession of Kerberos principals and passwords) would be less than optimal.

After three months of discussion regarding the functional requirements of each constituency for an institutional authentication/single sign-on project, the committee began the process of reviewing possible solutions. The group reviewed self-maintained solutions incorporating MIT's Kerberos version 5 and various available PKI components, as well as vended solutions from IBM, Platinum, CyberSafe, Tivoli, and Transarc/IntelliSoft. Some solutions were eliminated from consideration on the basis of their not meeting the committee's requirements for scalability or supportability (CyberSafe, Platinum AutoSecure). Other solutions were eliminated on the basis of their not providing the security enhancements desired in

conjunction with deploying an enterprise-wide single sign-on solution (IBM GSO, Tivoli User Management). Ultimately, the committee agreed to recommend a solution based on IntelliSoft's SnareWorks software.

At the time of this writing, the institutional single sign-on project at Duke is completing the final stages of the organizational funding review. Once a funding model is established for the project, work can begin in earnest on the wide deployment of authentication, security, and single sign-on features called for in the committee's recommendations at the institution.

Conclusions: Learning From Duke's Experience So Far

Clearly, implementing SSO for any but the smallest and most homogeneous of enterprises is a highly organization-specific task, and certainly no one can provide a true ''cook book'' for arriving at an agreeable and functional solution for use in all environments. Nevertheless, the authors are convinced that as organizational computing infrastructures become increasingly complex and users become more dissatisfied with the ease of use problems associated with heterogeneous computing environments, system administrators will increasingly be called upon to provide ''single sign-on'' solutions for their organizations. If the question hasn't been asked yet, rest assured that it will.

While no single set of rules can guide the administrator in designing a single sign-on solution for a particular organization, a few guidelines may be of assistance in the decision-making process:

- Any complete SSO solution must embrace not only critical applications, but also the operating systems they depend upon, the network services they rely on (ftp, electronic mail, etc.), and the underlying data services (HTTP services, database management services, etc.) required by applications. An SSO solution which supports only a handful of critical applications but does not encompass the underlying services those applications rely upon will be incomplete at best, and may well fail to serve the needs of end-users, not to mention the requirements of system administrators.
- In designing an SSO solution, weigh not only the cost of deploying SSO-related software and hardware, but also the cost of supporting the envisioned authentication and/or SSO infrastructure. Pay special attention to the ''hidden'' costs of re-engineering applications and operating systems to support any chosen SAR or SSO solution, and to the possibly-enormous costs associated with verifying user identities and (re)issuing authenticators to end-users. Frequently, these costs will far exceed the initial capital costs of deploying an SSO solution for an organization.
- Keep in mind any security policies already in place within the organization, and those which may be mandated by any given SSO solution. Duke, in particular, has suffered from a dearth of strong security policies, so enabling enforcement of stronger security policies was a significant factor in our investigations. Sites with established policies should take care to review the effects an SSO solution may have on the enforcement of existing policies before making any sort of SSO deployment decision.
- Decide initially which of the possible key features are of the greatest importance to the organization. We recommend consultation with both end-users and management to investigate the relative importance of four key features to the organization:
  - Reduced authenticators/sign-ons. If the primary goal within the organization is to eliminate the proliferation of authenticator information for end-users, a classical, vended SSO solution may be adequate. If the primary goal is to eliminate the proliferation of authenticators from the point of view of the system administrator, a SAR may suffice.
  - Site policy control. If the primary goal within the organization is to achieve tighter control over the enforcement of site security policies, a SAR may be the best solution, possibly with a vended ''key box''-style SSO solution.
  - Security enhancement. If the primary goal within the organization is to tighten security for specific applications and systems, a SAR approach using one of the more secure mechanisms discussed above may be ideal. In the case of Duke University, the additional security enhancements provided by SnareWorks' network encryption facilities were a significant factor in the choice of IntelliSoft's products. If security enhancement is of little or no concern, classical SSO solutions (such as IBM's GSO) may be most appropriate.
  - End-user convenience. If the primary goal within the organization is simplifying authentication interfaces presented to end-users, a targeted SSO solution, most likely without the deployment of a network-based SAR, may be the most appropriate solution. In these environments, organizations should consider the possibility of employing a ''niche'' SSO solution (CoreChange, for example, offers an SSO solution specifically targeted at the health-care market); such tightly-focused solutions can usually

provide end-users with features not available in more generic SSO solutions, but are frequently not applicable to the wide range of applications and systems for which large, unfocused organizations may need support.

- Plan from the outset for a much larger system than can possibly be envisioned. When the SSO project at Duke was first discussed, it was viewed as a relatively simple project supporting a few thousand users and a handful of mostly administrative systems. In the process of fleshing-out the details of various constituents' needs, we on the Duke project committee found that what was actually needed was an institution-wide infrastructure capable of supporting the 40,000+ current organizational affiliates, and capable of expanding to support a growing population of "new" affiliates associated with the University's ever-growing Duke Health Systems initiative. Building a system to be scalable can be difficult, but redressing a too-small solution once it has been deployed can be nearly impossible.
- Take advantage of any tools and resources already in place within the organization. At Duke, the prior existence of a large Kerberos version 4 realm has provided the institution with a "leg up" in developing an institutional SAR. While the decision to retain compatibility with the existing Kerberos version 4 realm did limit our choices with respect to SAR- and SSO-building products, it has enabled the institution to consider implementing an organization-wide solution without the initial costs associated with re-verifying 35,000 users' individual identities.
- Set reasonable, attainable goals for the SSO solution. Initial discussions at Duke centered on finding an SSO solution which would solve authentication and single sign-on problems for every application and environment in use at the institution: a full-scale single sign-on approach. Later, we came to realize that the goal of eliminating *all* user identity proliferation was too ambitious, and we have instead opted for developing a "reduced sign-on" solution. This has allowed us to provide for a higher level of functionality for those applications and systems most critical to the organization, at the expense of not supporting some less-critical applications and guaranteeing that some users of legacy applications will own multiple authenticators.
- Avoid implementation plans that introduce "flag days" for end-users. This is a lesson the authors learned in managing the growth and reorganization of institutional electronic mail infrastructures at Duke, and which applies at least as well to the deployment of SSO solutions within organizations. Flag days are extremely expensive in terms of goodwill capital with end-users, and in the case of SSO solutions, are likely to result in the sorts of unrecoverable failures which can require an entire organization to be recertified. A gradual, incremental approach to deploying an SSO solution can prevent organization-wide catastrophes in the event that unforeseen problems arise in the deployment of an enterprise-wide authentication mechanism.

A Note on Committee Dynamics: Staying Sane in a Political Minefield

If one factor can be expected to appear ubiquitously in the quest for organizational SAR and SSO solutions, it is politics. Having worked within the Duke computing infrastructure for a combined total of over 17 years, the authors expected political factors to interfere with the progress of Duke's institutional SSO project, yet even we were somewhat surprised at the extent to which the "SSO issue" was cause for political intrigue. Here are a few tips derived from our experience with the Duke SSO review committee:

- Take the time to involve a diverse cross-section of the community throughout the decision-making process.
  This was perhaps the most difficult and the most useful approach we used. We were fortunate at Duke, in that the request for a "single sign-on solution" came from relatively high on the corporate ladder (the Office of the Associate CIO). This facilitated our enlisting other computing professionals from within the central computing support groups at the University, as well as end-users, administrators, and managers from various departments around campus in our work. Although the diversity of the committee made our initial work slow (some members of the committee started with virtually no understanding of the issues involved), it paid off in the end with a broad base of support for the project, once it was fully described.
- Take the time to address first principles, and find common goals across the participants.
  In our case, this was a significant challenge. In particular, the authors found that with the exception of a few extremely technical staff virtually no one on the SSO review committee came into the process with a clear understanding of what "SSO" really meant. Many hours of meetings were necessary just to clarify the distinction between *authentication* and *authorization* for the committee (most members failed to see the distinction at first), and many more hours were necessary to work through the trade-offs between ease-of-use factors (of critical importance to end-users and managers) and security factors (of critical importance to

system administrators and programmers). In the end, more than half of the time spent in committee was spent discussing "first principles." Once these issues were agreed upon, however, vendor reviews and the choice of an approach were comparatively simple.

- Expect the process to take some time.
  At Duke, our original mandate was to complete the review of available options and return to the senior administration a proposal within three months. At the three-month mark, our committee had barely achieved agreement on the nature of the problem at hand, let alone selected a particular solution. The unfortunate absence of the authors' departmental director as a result of an extended illness allowed us to petition the administration for additional time. However, pressure mounted as scheduling of vendor presentations and meetings with the review committee and key managers outside the institutional computing infrastructure forced the project schedule to slip further and further.
- Expect the process to be expensive. Warn your management early.
  No matter what path an organization takes through the SAR/SSO obstacle course, be assured that the result will be more expensive than originally expected. If the solution(s) chosen are proprietary in nature, much of the expense may be in software licensing and acquisition. If the solution(s) chosen are "free," the costs may be weighted toward staff time for development and installation. Regardless, we urge system administrators working on SSO solutions for their organizations to be open and up-front with management about the real costs involved. At Duke, we expect the initial year of development and deployment to cost in the neighborhood of $1 million, with continuing costs between $100,000 and $200,000 per year. Original estimates of the total cost for an institutional SSO project were significantly lower, although compared to the costs of some of the major efforts ongoing at the institution (some of which will run into the tens of millions of dollars), the expense is less staggering. Note that Duke is a large and diverse enterprise, with more than 40,000 individual users. Your mileage may vary.

Software Availability

See Table 4 for URL references for software discussed in this paper.

Author Information

Michael Fleming Grubb has worked as a Unix system administrator at Duke University for six years. He is the principal architect of the campus-wide email and web infrastructure. He is also a licensed attorney. He can be reached via post at Box 90132, Duke University, Durham, NC 27708-0132, USA, or via email at <mg@duke.edu>.

Rob Carter has worked as a Unix system administrator at Duke University for over eleven years. He has been the lead system administrator for the University's public Unix computing facilities throughout their existence, although he did not invent Unix. He can be reached via post at Box 90132, Duke University, Durham, NC 27708-0132, USA, or via email at <rob@duke.edu>.

References

Bellovin, S., and M. Merritt. "Limitations of the Kerberos Authentication System." *ACM Computer Communications Review*, October 1990.

Dyer, S. "The Hesiod Name Server." *USENIX Conference Proceedings*, Winter 1988.

Fraser, B., ed. *RFC 2196: Site Security Handbook*, 1997.

Haller, N. and R. Atkinson. *RFC 1704: On Internet Authentication*, 1994.

Haller, N., "The S/Key One-time Password System." *Proceedings of the Symposium on Network & Distributed Systems Security*, Internet Society, San Diego, CA, February 1994.

Haller, N., C. Metz, P. Nesser, and M. Straw. *RFC 2289: A One-Time Password System*, 1998.

Hess, D., D. Safford, and U. Pooch. "A Unix Network Protocol Security Study: Network Information Service," *ACM Computer Communications Review* 22 (5), 1992.

Kaufman, C., R. Perlman, and M. Speciner. *Network Security: Private Communication in a Public World*. Prentice Hall, 1995.

Kohl, J., and C. Neuman. *RFC 1510: The Kerberos Network Authentication Service (V5)*, 1993.

Linn, J. *RFC 1964: The Kerberos Version 5 GSS-API Mechanism*, 1996.

Linn, J. RFC 2078: *General Security Service Application Program Interface*, Version 2, 1997.

Mockapetris, P. *RFC 1034: Domain Names – Concepts and Facilities*, 1987.

Mockapetris, P. *RFC 1035: Domain Names – Implementation and Specification*, 1987.

Mullan, S. *OSF RFC 92.0: DCE Interoperability with Kerberos*, 1996.

Pato, J. *OSF RFC 6.0: A Generic Interface for Extended Registry Attributes*, 1992.

Ramm, K. and M. Grubb. "Exu – A System for Secure Delegation of Authority on an Insecure Network." *Proceedings of the Ninth Systems Administration Conference (LISA IX)*, Monterey, CA, September 1995.

Ramsey, R. *All About Administering NIS+*. SunSoft Press, 1994.

Rigney, C., A. Rubens, W. Simpson, and S. Willens.

RFC 2138: Remote Authentication Dial In User Service (RADIUS), 1997.

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

Stern, Hal. *Managing NFS and NIS*. O'Reilly & Assocs., 1991.

Wray, J. *OSF RFC 5.2: GSS-API Extensions for DCE*, 1994.

Appendix 1: Tables

Application or System	Short-Term	Medium-Term	Long-Term
AFS	X	X	X
Unix net apps (ftp, telnet, etc.)	X	X	X
X11	X	X	X
IMAP4	X	X	X
POP3	X	X	X
HTTP/CGI	X	X	X
SAP R/3	X	X	
PeopleSoft	X	X	
LDAP	X	X	
Lotus Notes	X	X	
MVS/RACF	X		
Oracle8	X		
Netware (4+)	X		

Table 1: Systems and Applications Targeted for SAR/SSO at Duke.

Feature	Critical	Important	Nice-to-have
Strong Authentication	X		
Distributable Management	X		
Unix compatibility	X		
NT compatibility	X		
Standards Compliance	X		
Scalability	X		
Performance	X		
Network Security	X		
Stability/Easy maint.	X		
Win95/Win98 compat.		X	
MacOS support		X	
SSO features		X	
Application Support		X	
Published API		X	
SSO user interface			X
Automated user registration			X
Remote installation			X

Table 2: Decision Matrix: Features Required for SAR/SSO at Duke.

Vendor	Product	Strng Auth	Dstrb Mgmt	Unix Supp	NT Supp	Stds Compl	Scala blity	Perf .	Net Sec	Stab lity	Win 95	Mac OS
MIT	Hesiod	-	=	+	-	=	=	+	-	=	-	-
Sun	NIS	-	=	+	-	=	-	+	-	-	-	-
Sun	NIS+	-	+	=	-	-	=	=	=	-	-	-
---	S/Key	=	-	=	-	-	-	=	+	=	-	-
MIT	K4	+	=	+	=	+	+	+	=	+	=	=
MIT	K5	+	=	+	=	+	+	+	+	+	=	=
CyberSafe	---	+	=	+	=	=	+	?	+	=	=	-
IBM	GSO	=	+	=	+	=	+	?	=	+	+	-
Platinum	AutoSecure	-	+	=	+	-	=	?	-	?	+	-
IntelliSoft	SnareWorks	+	+	+	+	+	+	+	+	=	+	-

Vendor	Product	SSO features	Applic. Support	Pub API	SSO-GUI	User Regis.	Install/ Maint.
MIT	Hesiod	=	= (+/Unix)	+	-	-	=
Sun	NIS	=	= (+/Unix)	=	-	-	=
Sun	NIS+	=	- (+/Solaris)	-	-	-	-
---	S/Key	-	=	=	-	-	-
MIT	K4	+	=	+	-	-	=
MIT	K5	+	-	+	-	-	=
CyberSafe	----	+	=	-	=	-	?
IBM	GSO	+	+	-	+	=	+
Platinum	AutoSecure	+	=	-	+	=	?
IntelliSoft	SnareWorks	+	=	+	=	+	+

K E Y

+:	strength of product
=:	available, but not particular product strength
-:	not available or too weak to rely upon
?:	insufficient data gathered to determine

Table 3: Vendor Performance: Features of Vended Products.

Product	Location
Hesiod	ftp://athena-dist.mit.edu/pub/ATHENA/hesiod/
NIS	(Distributed with most Unix variants)
NIS+	http://www.sun.com/software/white-papers/wp-nisplus/
RADIUS	http://www.livingston.com/tech/docs/radius/
S/Key	ftp://ftp.bellcore.com/pub/nmh/
SecurID	http://www.securid.com/
MIT Kerberos V4	ftp://athena-dist.mit.edu/pub/kerberos/README.KRB4
MIT Kerberos V5	http://web.mit.edu/kerberos/www/
DCE	http://www.opengroup.org/dce/
KTH Kerberos V4	http://www.pdc.kth.se/kth-krb/
Tcl-Kerberos	http://www.neosoft.com/tcl/ftparchive/sorted/net/tcl-krb5/
Java-Kerberos	http://www.camb.opengroup.org/RI/www/jkrb/
GSO	http://www.software.ibm.com/enetwork/globalsignon
SnareWorks	http://www.isoft.com/
CoreChange	http://www.corechange.com/
Platinum AutoSecure	http://www.platinum.com/products/sysman/security.htm
Tivoli	http://www.tivoli.com/

Table 4: Software Availability.

Protocol	PSM	Authentication	Encryption	Authorization	SSO
Telnet, FTP	Yes	Yes	Yes	Yes	Yes
X11	Yes	Yes	Yes	Yes	N/A
SMTP	Yes	Yes	Yes	Yes	N/A
IMAP4	Yes	Yes	Yes	Yes	Yes
HTTP	Yes	Yes	Yes	Yes	Yes
POP3	Yes	Yes	Yes	Yes	Yes
R{login,sh}	Yes	Yes	Yes	Yes	Yes
NNTP	Yes	Yes	Yes	Yes	Yes
IIOPNS	Yes	Yes	Yes	Yes	Yes
SNMP	Yes	Yes	Yes	Yes	Yes
MQseries	Yes	Yes	Yes	Yes	Yes
PeopleSoft	Devel	Yes	Yes	Devel	Devel
SAP	Devel	Yes	Yes	Devel	Devel
LDAP	Yes	Yes	Yes	Yes	Yes
Lotus Notes	Devel*	Yes	Yes	Devel*	Devel*
SMB/Netbios	Yes	Yes	Yes	Yes	Yes
Oracle8	Yes	Yes	Yes	Yes	Yes
Arbitrary IP protocols	No**	Yes	Yes	No**	No**

K E Y

Yes:	SnareWorks provides this feature for the protocol
Devel:	SnareWorks is currently developing this feature
No:	SnareWorks does not provide this feature

N O T E S

*	Upcoming versions of Notes are reported to support X.509 certificate-based authentication. Such would make a Notes PSM redundant, since SnareWorks provides native support for X.509.
**	The SnareWorks API provides a mechanism for sites to develop their own PSMs, providing SSO and Authorization features for arbitrary IP protocols.

Table 5: SnareWorks Application Support Levels.