Global Information Assurance Certification Paper

Prepared by: Tom Schnittker
Assignment version: 1.2e
Date: 6/8/2001

Single Sign-On: A case study

Background

Working as a Security Administrator for a medium-sized hospital during 1999, our organization was faced with our current clinical systems being largely rendered obsolete by the specter of Y2K. The current system was mainly a single mainframe application that provided a variety of applications. The system was had been in place for well over a decade and was largely based on seriously outdated technology. While it would have been possible to make the application Y2K compliant, the expense and time it would have taken were hard to justify and in the end it would have still been based on outdated technology. Based on this the decision was made to replace the system. After considerable analysis, the decision was made to replace each of the clinical functions provided by the old system with a variety of "best of breed" applications rather than with a single application or a suite of products provided by a single vendor.

This decision put several hurdles in our path. The first was the problem of logins. Under the old system, our users were only required to enter a single user ID and password to access the various clinical functions that they needed to access. Under the new systems, a user would be faced with up to ten individual user IDs and passwords in the worse case scenario and in most cases each user would be faced with a minimum of three or four user IDs and passwords to remember. This also meant from an administrative standpoint instead of entering each new user into a single system as we did under the old system, we would be faced with entering each user into a number of different security databases.

Secondly, the applications presented several security issues. In determining the applications that would be used, the functionality that the application provided to the user was given great consideration, while the security features of the applications were given little if any consideration. This meant that there were a variety of security features spread across the applications. Some of the applications supported expiring passwords and some did not. Some of the applications provided strong controls for passwords such as being able to set minimum length, preventing reuse, and rules for password content while others did not. The applications themselves also ranged from a random mix of GUI-based client-server applications to telnet-based applications.

These circumstances made us realize that we would be putting an unrealistic burden on the user community to remember a large number of passwords and IDs. While these IDs and passwords might all start out as the same thing on day one, over time they would become increasingly out of sync as time went on,

increasing the burden even more. This combined with the lack of adequate controls in many of the applications virtually guaranteed that our users would choose weak passwords. This proliferation of IDs and passwords also pretty much guaranteed that we would see the number of calls to our support line to reset forgotten passwords would go through the roof.

So faced with these issues the decision was made to investigate the possibility of implementing a single sign-on solution to alleviate many of the burdens and hopefully strengthen some of the weakness that were in the security features of many of the applications.

What is Single Sign-On (SSO)

No matter what the implementation, the overall goal of single sign-on is the same. It is to provide a secure method of authenticating a user one time within an environment and using that single authentication as a basis for access to other applications or operating systems. SSO can be carried out in a variety of methods, ranging from complex to simple.

In its most complex form, which is true SSO, all the applications share a common security mechanism that allows a user to truly have a single user ID and password to access all resources. This is very attractive from an administrative standpoint because it means user accounts can be created in a single database that all applications access for authentication. However, this method is difficult to implement across an environment that consists of multiple vendors all insisting on using proprietary security schemes. Based on the fact that our application mix was spread across a variety of vendors, all using proprietary security schemes, it was obvious that this was not going to be an option for us.

In its simplest form, the term SSO is actually a misnomer. In this instance each user still has multiple user IDs and passwords, they are simply shielded from the user in such a way that they do not know that they are there. In this instance, a database or directory is used to store credentials for each application that a user needs to access. This database is housed on an authentication server. A client application of some type then functions as an intermediary for the user between the user application and the database on the authentication server. For instance User A has access to App1, App2, and App3. Within the SSO database, a record for User A would exist that might appear similar to the following:

User ID	App Name	App ID	App PW
UserA			
	App1	UserA	summer
	App2	UserA	football
	App2	45532	032364

As you can see User A still has a separate set of credentials for each application and that it is not even necessary for these credentials to match.

When User A first authenticates to the network for the day, the SSO client uses that authentication to also authenticate the user to the authentication server. When User A then starts App1, it is sensed by the client which then retrieves the credentials for App1 from the database and supplies them to the application. The method for presenting the credentials to the application varies. Some applications use an API that actually hooks into the applications program to supply the credentials directly based on programming calls. This method is quite seamless and the user probably doesn't even know that the authentication happened. This method tends to be somewhat invasive in that it may alter or replace code within the application and may have implications for support agreements with the vendor of the targeted application.

Other simple solutions monitor what is happening within the application window and sense when the application is waiting for a user to supply a user ID and password and passes along the credentials at the appropriate time. This often involves writing a script for the application to deal with what is happening on the screen. This can become quite complex due to the need to account for virtually every possibility of what might occur on the screen such as logon prompts, password expiration notification, incorrect passwords or user IDs. This method also tends to be quite flexible and non-invasive with an ability to deal with all types of applications ranging from GUI-based applications, telnet-type applications, or browser-based applications.

Our choice

In the end we chose to implement a solution that fell into the latter category. It used a central database of user credentials housed on an authentication server. It was able to utilize our primary method of authentication, which was through a Novell client running on NT workstations to authenticate the user to the authentication server. It also used a non-invasive method of using a scripting language to monitor the execution of the target application on the workstation and supply the appropriate credentials when needed.

Pros and Cons

In reaching our final decision, a number of pros and cons were identified. The pros took care of themselves and the cons were identified and policies were developed to minimize the exposures they presented it.

The most obvious plus for the system was the fact that the burden on our users to remember a large number of user IDs and passwords would be largely eliminated. Due to the fact SSO was implemented simultaneously with the new clinical applications, our users probably do not even have a true appreciation for

what SSO is doing for them because they never had to remember a single password for any of them for even as much as a day. Our support line also never had to deal with the onslaught of password related calls they would have received without SSO in place.

We also received several other benefits that were transparent to the user but were greatly appreciated by IT. The SSO solution that was selected allowed us to not only store information on the credentials of each of the users within our environment, but it also allowed us to create policies for each of the applications that would interact with SSO. This allowed us to work around many of the limitations that existed in the security weaknesses of many of the applications. For instance, we could create a policy that would cause SSO to automatically generate a new password every X number of days and the length and content of the password. This accrued several benefits for us:

- Stronger passwords are generated
- Passwords are expired more frequently than what would otherwise be tolerated by the user
- Since SSO was generating password and supplying it to the application, in most instances the user doesn't even know what the password is. This eliminates any possibility of the password being written down or shared with others

The ability to take advantage of these benefits varied from application to application. If an application did not support expiring passwords, we could have the policy for the application generate a new password on a frequent basis. However, if the application did provide a method for changing the password, there was no way to create logic within the script to pass the new password to the application. Within the policy we were also capable of having SSO generate new passwords that were of significant length and consisting of a random mix of alphanumeric and special characters. However, some of our applications would only allow passwords of a limited length and character set. Therefore the security features provided by SSO were still somewhat bound by the capabilities of the application. The application policies also provide the ability to limit the time of day that the application is available.

Another benefit that was derived from having all of a user's application credentials stored in a single location was the rapidity with which we could remove their access to a large number of applications. Without SSO, if it became necessary to block a user's access to a number of applications, it would have been necessary to go into the user database for each application and eliminate or disable the user's access. Under SSO, disabling or deleting the user's account on the authentication server effectively eliminates their ability to access all of the applications for which they had access.

Beyond the positives of SSO, it was also necessary to consider several potential negatives associated with its implementation. Considering that a user's access to a variety of applications is now governed by a single user ID and password, it becomes even more critical that the sanctity of that single ID and password be protected.

As stated by Fred Trickey, in Secure Single Sign-On: Fantasy or Reality:

> "One problem with any form of single sign-on that is of great concern to many security practitioners is that a single ID and reusable password would become the 'key to the kingdom.' If a user's password were compromised, the intruder would have access to any and all applications to which that user was authorized. Single sign-on, therefore, increases the need for robust, secure authentication: fully encrypted logon at a minimum, or the use of one-time passwords, challenge response systems or biometric authentication."

An even greater concern than weak authentication methods with SSO in the environment, is simply situations where users walk off and leave a workstation that they are logged into unattended. Anyone walking up and accessing an unsecured workstation would have access to any of the applications that the currently authenticated user has access to. This can be guarded against in a number of ways. Many SSO client applications provide methods for timing out a session after a period of inactivity. The user would then have to re-authenticate to the SSO server before running any applications. User education also becomes a priority in ensuring that the user community employs sound practices in making sure they do not leave unattended workstations in a vulnerable state.

Another major concern is the authentication server and its database. Considering this is a single repository for all of the user IDs and passwords for every application in the environment that is SSO-enabled, this box becomes a prime target for anyone trying to gain unauthorized access. This makes it critical to ensure that the SSO application implements strong methodologies for protecting the database. It is also critical to ensure that the transport mechanism that is used to transmit the user's authentication credentials between the client and the authentication server is secure and not susceptible to sniffing.

Having all of your users' credentials stored in a central location also presents some interesting opportunities for disaster. Given a large environment with thousands of users and a large number of mission-critical applications enabled for SSO, what happens if the SSO server crashes? It is imperative that system redundancy be built in to the SSO environment. This includes multiple servers that can fail over automatically as well as redundant pathways to each of the servers. The SSO application should also provide support for the automatic synchronization of the database across the redundant hardware. Our

configuration provided for two servers each with two network interfaces. Each interface is connected to a different segment. The DNS entries for the servers are configured to provide automatic round-robin access for each of the interfaces. The client is configured with a primary and secondary address to contact the server. If a successful connection is not made to the primary address and attempt will be made to contact the second interface. This provides load balancing between the servers as well as automatic fail-over should anyone of the servers or their individual interfaces fail.

Conclusions

With almost two years having passed since our initial implementation of SSO, I can't imagine what life in our environment would have been like without it. Beyond the fact that a large percentage of IT would have been lynched by the hospital staff had we put them in a position of remembering more than a couple of passwords, SSO has paid many other dividends. SSO has helped us overcome the security weaknesses of many of the applications that continue to proliferate within our environment by allowing us to implement expiring passwords and strong passwords for those applications that do not support them. Of the negative aspects that may be associated with SSO, I have yet to find one that can not be over come. As with any security vulnerability, it is simply a matter of identifying what that vulnerability is and then implementing the appropriate policies and procedures to eliminate the vulnerability or reduce it to an acceptable level.

References

- "Single Sign-on dangles prospect of lower help desk costs" http://www.infoworld.com/articles/es/xml/00/10/02/001002esnsso.xml

- "Secure Single Sign-On: Fantasy or Reality? The 1997 CSI Single Sign-On Product Matrix", Fred Trickey, Columbia University, CSI Advisory Council http://www.gocsi.com/sso_ft.htm

- "Understanding Single Sign-On", Information Systems Audit and Control Association http://www.isaca.org.za/H_sso.htm

- "Single Sign-on", Pete Loshin, Computerworld, 2/5/2001 http://www.computerwrold.com/cwi/story/0,1199,NAV65-663_STO57285_NLTs,00.html

- "Secure Single Sign-On? Dream on", Fred Trickey, Information Security Magazine, 1998 http://www.infosecuritymag.com/articles/1998/septSSO.shtml