Internet2 Middleware WebISO
Document Version: Draft 02

Authored by: Chad La Joie (clajoie@vt.edu)
Edited by: Russell Tokuyama (russ@hawaii.edu)

Date Submitted: 7/28/2002

Trusted Delegation of Privileges in an N-Tier Environment

Abstract

When people consider web based initial sign on (WebISO) systems, such as those described by the WebISO group, one of the problems they are often trying to solve is how to allow one principal (be is a person or a system), to work on behalf of another. This delegation of privileges, which this document focuses on, is a different problem domain than that of a "pure" WebISO system. This document will being with a discussion of the basic concepts, vocabulary, and mechanisms of privilege delegation systems and how it differs from WebISO systems. Then an analysis of the security risks associated with the different delegation mechanism will be presented. Following that case studies of different delegation systems, some of which incorporate WebISO systems as well, will be given. Then some considerations for vendors and developers who wish to have their software work well with delegation systems is presented. Finally some references and related reading material as well as acknowledgements are given.

Table of Contents

1. The Basics of Privilege Delegation Systems

1.1 Introduction

In numerous applications it is sometimes necessary to "become" some one else, or some other principal in order to take advantage of some privilege(s) they have. Unix systems have had this concept for decades in the *su* command. With the advent of web based enterprise applications it became clear that something similar was needed for them. That is it became necessary to be able to pass around a set of privileges in some manner. Since privileges are normally associated with a principal (very often a person) the problem became a question of how to pass around these principal's identities and how to allow one system to use that identity to gain access to information on another system.

Those systems which use a principal's identity to access another system is known as a delegate. The system which is being accessed is known as the destination system. The following example demonstrates these abstract ideas. Assume that a university has a portal and a web based email system. A user logs into the portal and goes to a page that informs the user that they have five new email messages. The user clicks on a "Read your Email" link and is presented with their inbox in the web mail application. In this example the portal acted as a delegate for the user to the web mail system.

1.2 Tiers

The example described above is known as a 3-Tier system as there are three principals or systems involved; the user's browser, the portal, and the web mail system. This represents the simplest form of a privilege delegation system. Another more generalized form of delegation system is known a N-Tier. In this model each destination system can also act as a delegate. In this manner a chain of systems can be created. The portal/webmail example presented above can be further extended to express this model. Assume that one of the emails the user received was a request to attend a specific meeting inside of which was a link to the university's calendaring application which the user can use to accept of decline the invitation. The user clicks on the link and is presented with her calendar and the option to accept or decline the meeting.

In this expanded example there are 4 principals or systems; the user's browser, the portal, the webmail system, and the calendaring system. This would be an example of the N-Tier model (in this case N is 4). More complex examples could be presented but they would simply add more delegate and destination systems. In theory this chain of systems could continue until it touched every web based systems at your university, and perhaps even systems outside of your university.

1.3 Mechanisms

Delegate frameworks usually use one, or more, of the following four methods to assume the identity or privileges of a user when working with the destination systems. Those four methods, masquerading, trust, backdoor, and proxy credentials, are described below. Different privilege delegation frameworks will support different combinations of these mechanisms. Some will support only one whereas others, which try to address a wider range problem areas, will support more then one. See section 2 for information on the security issues with each of these mechanism.

1.3.1 Masquerading

Delegate systems which use the masquerading method store the user’s credentials when initially presented with them. When the user attempts to go to a specific destination system the delegate system presents the user's credentials to the destination as if it were the user, in other words the delegate system masquerades as the user to the destination system. Once the delegate system has accessed the destination system as the user it can perform any of the actions the user is permitted. This method is relatively easy to implement, assuming that the destination system does not have a principal id for this user that is different from the one the delegate system received and that the destination system has a well documented API.

1.3.2 Trust

In the trust method, the delegate system identifies itself directly to the destination systems, perhaps via some other authentication process. Assuming a trust relationship has been set up between the delegate and destination systems, the destination system will allow the delegate system to act as, or request information for, any given principal. Some more advanced trust mechanism implementations may require the delegate system to prove in some manner that it has, either explicitly or implicitly, received the principal's permission to access their information or act on their behalf. This method can also be easy to implement, assuming that both the delegate and destination systems have either a well document API or source code available.

1.3.3 Backdoor

With the backdoor method the delegate system completely circumvents the destination system's API and directly accesses its data store in order to retrieve the information necessary. If the user is expected to interact with the destination system the delegating system will usually setup a session for them in the destination system and then redirect the user to the destination. This model requires that the delegating system have intimate knowledge of destination system.

1.3.4 Proxied Credentials

In this method, a delegate system requests a proxy of the user's credentials from a centralized authentication/authorization system. These proxied credentials, or tickets as they are often called, are normally created such that only the central authentication system can resolve them into information about the user. Once the delegate system has a ticket, it then presents the ticket to the destination system as a means of validating the user to the destination system. Usually, this requires that the destination system present the ticket to the centralized authentication system which checks the validity of the ticket, resolves the it to the user's id, and presents the id back to the destination system so that it may create a session for the user. Some implementations also allow the destination systems to accept or reject proxied credentials on the basis of which delegate system is presenting the credential. For example, a web based mail client may accept proxy credentials from the portal application but not from the student information system. Like the backdoor and trust methods this one requires a good deal of knowledge about both the delegate and the destination systems as it is likely that the authentication portions of each will need to be reworked to take advantage of this method.

1.4 Destination Systems

Each of the above techniques was created to deal with distinct types of a destination systems. The three most common types of destination systems are open, extensible, and blackbox.

Open

Open systems are those systems for which the source code is available and can be modified. These systems lend themselves best to the backdoor and proxied credential approach, as it’s possible to understand precisely how the system's internals work. Since the source code is provided these systems are the most able to be integrated with a privilege delegation frameworks.

Extensible

Extensible systems are those for which the source code is not available but hooks, usually provided by an API and library, allow control over certain aspects, if not all aspects, of the authentication/authorization process. These types of systems lend themselves well to the masquerading and proxied credentials methods, but may also be used with the trust method. These systems also have a high degree of success in being integrated into an privilege delegation framework.

Blackbox

Blackbox systems are those systems which neither provide source code nor a means to control aspects of the authentication/authorization process. Most off-the-shelf software applications fit this category. The masquerading method works well for systems that provide an easy way to deliver credentials (for instance, a POST to a particular CGI). Often, however, the backdoor method is required. This approach is often arduous and time consuming as it requires that you spend large amounts of time probing the internals of the system and learning how the system works, usually, with little or no help from the vendor. Due to their closed nature, these systems are the least likely to be integrated into a privilege delegation framework.

1.5 How WebISO and Privilege Delegation Systems work together

While it helps to think of WebISO and privilege delegation systems as logically different systems, they are almost always implemented within a single software package. It may however be of benifit to look at the differences of these two systems before discussing how they work together. First, pure WebISO systems are designed to only be used with web applications whereas privilege delegation systems are meant to work with any system which has the concept of user based access control (though in practice most privelege delegation systems focus on web applications). Second, WebISO systems are focused on providing applications with assurance that a user is who they say they are, that is, that they have been authenticated. Most delegation systems are not so much concerned with this aspect as they are with the ability for one system to use a principal's privileges on the destination system, though most delegation systems do take into account some level of authenticity checks. Third, WebISO is concerned with the security restrictions that may exist between web applications, for example if one web application requires a userid and password for login and another requires smartcard and pin for login, the WebISO system must be able to understand the difference in these two security realms. Again, delegation systems aren't so much concerned with how the person is authenticated, or supposed to be authenticated, between applications, it is instead concerned with the person's access to restricted resources/functions.

So how do WebISO and privilege delegation systems work together? The following example with help frame the discussion on this. Suppose a user is required to enter a user id and password to access you portal system. In order to display all the information the university would like to in the portal it must have access to both the user's email account (to get the number of new messages) and the user's enterprise information system account (to get thier list of classes). The mail system requires a

Kerberos id (not the same as the portal user id) and a ticket to access it while the enterprise information system requires a numerical PIN and password to access records. Assuming you wanted to integrate all these systems you would have at least two issues (probably more) to overcome. First, how do you take the portal user id and translate it into the principal ids required by the other systems. Second, how do you access the others systems, on behalf of the user, to get the information you need.

As mentioned above privilege delegation framekworks provide the answer to the second question. Through thier numerous mechanism they can access the destination systems either directly as the user by masquerading them or using some proxied credentials or inderictly by using a backdoor or an established trust relationship. A WebISO system provides the means to answer the first question. It's concern with user authenticity, especially if it supports the concept of security realms, means that it's well suited to translate one principal id to another. If your privilege delegation system can interoperate with your WebISO systems, and it's likely that it can, then the delegation framework can simply request the translated ids as it needs. Once it has the ideas it can employ any of the mechanisms mentioned above to gain access to the user's information on the destination systems. If the method choosen is the use of proxied credentials it's likely that WebISO will be the one to construct the credentials used by the delegation framework.

2. Hello ease of use... good bye security?

With the ability to delegate user privileges to other systems the question of security should be in the forefront of everyone's thought. This section looks at some of the security risks associated with this framework in general as well as those security risks inherent to each method of delegation.

2.1 General Security Risks

Like most authentication/authorization systems it's important to know who you're talking to. If you are not checking the identity of all the systems involved (delegate checking the destination system's id, etc.) then you're leaving yourself open to a man in the middle attack. Also, this check must require something stronger than just an IP address check or some other easily spoofed set of information. Instead things such as client cert verification with SSL connections can be used. Other, simpler to implement, means might include things like having the delegate system "log into" to the destination system. That is, the delegate system presents a userid/password pair of its own.

In addition to verifying the identity of those applications that are being communicated with, there is the need to ensure that no one else is listening in on the conversation. If your delegate system and all the applications it uses are not using a secure network then it should assumed there are hostile applications listening to all your traffic. Securing your traffic can easily be done by using SSL assuming your delegate system uses protocols that can be tunneled over SSL. If it does not then each system, the delegate and destination, should encrypt and decrypt each message as they send and receive.

A delegation framework's handling of sessions on the different destination systems can also be the cause of a potential security risk. When a delegate system accesses a destination system it may create a session which must be managed in the same fashion as any other user session. Normally a session would end when a user issued an explicit log out command to the application, but because of the nature of delegating frameworks the user can not issue this command to all the destination systems for which a delegate system may have created a session on. Many delegating frameworks have an ability known as cascading logouts where the user's explicit log out command issued on one system is sent to all the destination systems on which the user has a current session. If this process isn't handled correctly, or isn't done at all, session may remain open on the destination and run the risk of being hijacked. This alone could be disasterous if, for example, the attacker managed to hijack a privelaged account on your enterprise system, however the risk is compounded by the fact that the attacker may be able to use this session to create sessions on other systems by employing the delegation framework.

2.2 Risks with the Masquerading method

If your delegate systems uses the masquerading method, one of the biggest security threats to be aware of lies with the delegate systems. In order to masquerade as the user these systems must store the user's principal and credentials somewhere. If this location is in a file on the disk then some one who compromises the system may be able to read the file and harvest the information in the file. Another common place to store this sort of information is in the memory where it's "safer". The problem with this is that by default many applications are compiled with debugging information which numerous tools use to read memory that the application is using. Java, for instance, provides a very nice API that gives access into the JVM's memory space; thus, allowing debugging tools to do just that. Again, if the system has been compromised the attacker could most likely easily install these tools and read the information right from memory.

Most applications that use the Masquerading method often claim to protect against this threat by encrypting the information when they write it to file or store it in memory. The fallacy here, however, is that this offers additional security. In order for the delegate system to read the encrypted data it must have a way to decrypt it. If the application has access to the key needed to decrypt then the attacker that compromised the system does as well. The only additional security that is gained by encrypting the information is that the attacker now needs to figure out which encryption method is being used. Once that's done she will be able to read the information just as if it was in plain text.

Unfortunately there is no good way to solve these problems. Some programming languages have features which help; Java's sand boxing features for instance, however, none of them offer fool-proof solutions. For these reasons, the masquerading method is not recommended for delegate systems where security is of great concern.

2.3 Risks with the Trust method

The major problem with the trust method is the fact that the delegate system's access to the destination system is entirely based on the identity of the delegating system. As was discussed in the general risk section this can be a problem if the destination system simply takes the delegate system at its word or only checks information that is easily spoofable (MAC address for example). As discussed above, probably the most secure solution to this problem is to make sure that both delegate and destination systems use SSL to communicate with each other and enable client certificate verification. However, this adds a lot of overhead to the systems, especially during the initial setup of the connection. Some of this can be overcome with the use of a persistent connection, but there is still a going to some overhead for maintaining that.

Another risk associated with the trust method is administrator negligence. If a system in the trust network is compromised, it is possible that the administrator of the compromised system may not inform the administrators of the other systems in the delegation framework. If that occurs then the attacker may now have access to all the other systems in the trust network. This can be taken care of by good communications and, if your delegation framework has the ability, revoking trusted access from the compromised system.

2.4 Risks with the Backdoor method

The biggest threat inherent in the backdoor method is ignorance. If the backdoor method is used and the person implementing it does not have a firm grasp on the inner workings of the system it is possible they could open up security holes. For instance, referring back to the first general risk, if the destination system doesn't check the identity of the other system then any external system may be able to gain access to the destination system.

2.5 Risks with the Proxied Credentials method

This is probably the most secure of all the methods discussed here. Assuming that all the general risks have been addressed there is only one other issue to consider. Each ticket that is issued by the central authentication/authorization system should be unique and have a limited lifetime in order to ensure that replay attacks can't be launched immediately or in the future should an attacker get access to that ticket.

3. Case Studies

This section is still in progress, however, here are some links to the systems currently planned for case study review.

3.1 Duke University's WebAuth System

Duke's WebAuth site.

3.2 Virginia Tech's authPortal System

A couple of papers on authPortal.
AuthPortal, version 3, overview
Towards a WebISO Model

3.3 Yale's Central Authentication System (CAS)

Yale's CAS site.

4. Considerations for Application Vendors/Developers

[in progress]

5. References and Related Reading

[in progress]

6. Acknowledgements

[in progress]

G. Glossary

[in progress]