Secure Single Sign-On using CAS and OpenID

Eng. Mihai-Gabriel IONIȚĂ  
Faculty of Military Information and Electronic Systems  
Military Technical Academy  
ROMANIA  
mihai_ionita01@hotmail.com

Abstract: As the number of connected applications increases, user convenience makes them use the same password, which can create vulnerabilities in an organization’s infrastructure. Therefore, those with security concerns regarding these activities, conducted in the electronic environment, came to support the idea of Single Sign-On systems, having the user in mind. This concept facilitates the authentication process to various resources available on the Internet, or locally, using a single pair of credentials, only once, so it makes the „one for all” password, the method of writing passwords on post-it’s glued to the monitor, or keeping them stored in unsecured text files, sent into oblivion. This paper aims to make a comprehensive analysis of the infrastructure of security and authentication systems on the market. On the other hand, the project presents the implementation of several open source solutions such as OpenID, NTLM in PHP or the Central Authentication Service. For the application that uses OpenID an online store that has an administration page, to which access is restricted using OpenID credentials, was further secured by filtering these IDs in a MySQL database to prevent access to anyone having a valid OpenID credential. For the demonstration of NTLM enabled SSO, another website is used in which access has been blocked to a page. For accessing the page, a valid Windows authentication credential is required. Both the online-shop and demonstration site for NTLM are written in PHP. In addition, the systems using them are implemented also using PHP. For implementing the CAS demo, two Tomcat servlets have been used for demonstration purposes, additional filtering being done by an ApacheDS server.

Key-Words: Single Sign-On, Security, Authentication, CAS, OpenID, NTLM, PHP, MySQL, ApacheDS

1. Introduction

Security, in IT terms, should not be opposed to ease of access. The Single Sign-On concept is part of the few IT technologies that give the common user security, if used correctly, and ease of access to the resources he needs to fulfill his work. SSO is very important when it comes to securing the applications used daily. SSO is being implemented everywhere around us. It can be said it is a common technology if we look at Microsoft’s, Google’s or Yahoo’s universe. All the resources available on any of the above-mentioned “platforms” are accessible via a unique pair of username and password. The uniqueness of this credential makes a SSO implementation an important single point of failure. If an attacker seizes control of the server, or makes it unreachable, unresponsive, he blocks the whole activity of the attacked company. Therefore, being critical for an enterprise, this is why the security of such a platform has to be taken into discussion. On the other hand using the same password for every resource accessed daily, in the absence of SSO makes the user equally vulnerable if not even more vulnerable. Hacking the weakest “link” from the site chain that is using the same credentials, and getting access to those credentials makes the user equally exposed as hacking the SSO server. Nevertheless, the SSO server, or servers, should be far better protected than any usual site, thus increasing the user’s security. Even though the benefits of SSO include the following:

- A single login reduces phishing success, because users are not used to entering their password without thinking.
- Helps reduce password fatigue caused by using different username and complex password combinations.
- Reduces the time spent re-entering passwords for the same identity.
- It is able to support conventional authentication such as Windows domain credentials.
Facilitates centralized reporting for compliance adhering.

There are different critics about this security model because as it was said before it represents a single point of failure. Therefore, SSO requires an increase in measures for protecting user credentials, and should be combined with strong authentication methods, such as smart cards and OTP tokens. (1)

1.1. Common SSO Configurations

Kerberos based SSO initially prompts the user for credentials and generates a TGT. Additional applications to which the user is authenticating use this TGT to acquire an ST for proving the user’s identity to the resource he is trying to access, without prompting the user for credentials.

The smart card base approach uses the smart card every time the user needs to authenticate, by querying the smart card for the certificate stored on it, without prompting the user for the PIN, but once.

The OTP token model uses a two-factor authentication with one-time passwords, which is the industry’s best practice for the time being because it prohibits unauthorized access to protected resources.

The integrated Windows Authentication uses SPNEGO and NTLMSSP authentication protocols, commonly used for automatically authenticating connections between IIS servers and Internet Explorer. (1)

Bellow, in figure 1, by the means of SSO based on inter-domain trust, a user is enabled access to multiple resources across multiple, separate, domains, with only one set of credentials.

With the expansion of SaaS (Software as a Service), and social platforms, more and more content providers deploy WEB-SSO schemes to facilitate users access to the resources they provide, this way getting the edge on competition. This is where the problem arises. The key word in these implementations is convenience, and not security, as it should be. An interesting paper (3) analyzes the complexity brought in WSSO schemes by rich browser elements such as Flash or scripts.

2. Security concerns regarding different implementations of SSO

The authors have discovered eight serious logic flaws by analyzing the traffic between browser and server, to recover important semantic information for potential exploit opportunities. These opportunities lead to the discovery of real flaws. These vulnerabilities permit the attacker to impersonate a victim.

2.2. OpenID

OpenID is an open standard and promotes authentication in a decentralized manner. It is also, one of the pioneers of SSO implementations. It is very flexible, allowing the use of credentials such as username and password pairs, all the way up to biometrics and smart cards. It is also widely used by large enterprises such as AOL, BBC, Google, IBM, MySpace, Orange, PayPal, VeriSign, Steam and Yahoo, to name only the biggest. (4)

It was reported that there were over one billion OpenID enabled user accounts and 9 million websites using OpenID as of December 2009. (5) In addition, as anything implemented on a large scale in the IT&C industry, it becomes
“interesting” for attacks of impersonation. The security issues with this protocol are present in the framework for managing the communication that takes place between the IdP, identity provider, and the relying party, or acceptor. Despite the protocol’s trivial aspect, from a user’s point of view, seven transactions take place in the background. Every one of these steps can be exploited in some way; this is why the implementation has to be secured against the following.

When a user logs into a site and provides their login, the relying party, has to download the URL and extract the identity provider’s address to continue. It is trivial to specify an arbitrary port, for example: http://www.mta.ro:1/, and cause the relying party to port scan any host on the internet, for the attacker, without any trace back to him. The attacker can trick the relying party into accessing scripts from an internal host, otherwise inaccessible for an external attacker, with carrying unauthorized actions in mind by submitting the following URL: https://192.168.10.120/internal/auth?ip=1.1.1.1. Similarly, the acceptor has to protect itself from an attacker trying to bypass the firewall by forcing the web server to connect to its un-firewalled ports using a link like this one: http://localhost:8080/. As it was said before, denial of service attacks (DoS) are critical for an SSO server. Therefore, the amount of time and data, a single request is allowed to consume has to be limited. Alternatively, an URL like http://www.youtube.com/largemovie.mpg could crash a server. At the same time not only a movie, but also a script with an infinite loop, for example, could be used to deny access to the server for legitimate users; www.mysite.com/infinite-loop.sh. In addition, for stopping the download from malicious URLs any downloads should be limited to https or http. Other protocols such as FILE or FTP should be explicitly disallowed. Otherwise, an URL like file:///dev/null could be used to crash the server, or using a link similar to file:///file/malicious.txt could be used to infect the server with a computer virus. (5)

Figure 2. Diffie - Hellman exchange

In figure 2, there is a caption of a Diffie-Hellman key exchange necessary for guarantying data integrity through HMAC. Unfortunately, DH is vulnerable to man in the middle attacks. Therefore, it is advised to use HTTPS. After identifying the IdP and successfully exchanging keys, the RP redirects the user to the IdP for actual authentication. This redirection is made through a simple HTTP redirect construct. The RP specifies itself the IdP’s address. Thus, a malicious RP can easily redirect a user to a malicious, identically looking provider, with the goal of stealing the user’s credentials. Phishing is the main attack method against this protocol. Worth mentioning is that even a maliciously formed URL could send the user to a cloned site for stealing his credentials. For example, the string http://IonitaM.myopenid.com/ could return

<link rel="openid.server" href="http://www.myBAD.com/server"/>

instead of

<link rel="openid.server" href="http://www.myopenid.com/server"/>

Again sending the user to a malicious IdP, designed for phishing credentials. Trustworthiness of an IdP cannot be emphasized enough. After the user logs in and is authorized by the RP, he can be logged in automatically into other sites supporting OpenID. Thus, a malicious IdP can easily spy on the user’s activity. With the current architecture of the protocol, there is little the user can do against being spied upon by rogue or even legitimate IdPs, except for using...
multiple logins, which would reduce the benefits of the protocol. (5)
Once a user is authenticated, the IdP redirects him back to the original web site, updating the successful login status through this type of URL:

```
Location: http://www.somesite.com/finish_auth.php?
openid.assoc_handle=%7B6HMAC-HA1%7B477bb..&
openid.identity=http%3A%2F%2Fjohn.doe.name%2F&
openid.mode=id_res&
openid.return_to=http%3A%2F%2Fwww.somesite.com&
openid.sig=vbuYN6n398e81kz119t830%2F4%3D&
openid_signed=mode%2Cidentity%2Creturn_to&
nonce=WV8o75KH
```

**Figure 3. Redirect to RP URL**

The problem with this URL is that anybody sniffing the network, can obtain this URL, and replay it for getting logged into the site in the name of the victim. Because of the nonce, the replay attack works only if the attacker is the first to provide this URL. Moreover, this actually is not that hard, if the attacker controls the network, he can reset the victim’s connection after getting the URL. Another type of attack that the implementation of OpenID should be wary about are the XSS type, Cross-Site scripting ones.

```
<iframe id="login"
src="http://bank.com/login?openid_url=John.doe.name" width="0"
height="0"></iframe>
<iframe id="transfer"
src="http://bank.com/transfer_money?amount=100&to=attacker"
width="0" height="0"></iframe>
```

**Figure 4. Cross-Site Request Forgery**

The above HTML code contains two hidden iframes, which silently login the user to their banking account and transfer money to an attacker. The attack succeeds because the IdP dictates the user login security policy. (5)

### 2.2. Central Authentication Service

Shawn Bayern from Yale University conceived CAS. Starting from 2004 it is maintained and developed by JA-SIG. When a user tries to access a resource protected by a CAS server, he is redirected to the CAS login page. The CAS server authenticates his credentials to a database: MySQL or LDAP, the latter being the case in the demo implementation for this paper. If authenticating the user is successful, the server redirects the user to the initial application, alongside an ST, security ticket. The application validates this ticket by contacting the CAS server through a secure connection, HTTPS, in the current demo application. For this verification to be successful, the page that hosts the requested resource has to present to the server its own, valid, service identifier. If this verification succeeds, the CAS server offers the application the requested, reliable, information about the user’s login. (7)

For integrating the described SSO capabilities, CAS uses a series of URIs and cookies, as follows:

- `/login` – Credential receptor or requestor.
- `/logout` – Destroys a user’s SSO session through the cookie containing the JSessionID.
- `/validate` – Verifies the validity of a given ST.
- `/serviceValidate` - Verifies the validity of a given SI, service identifier. It returns an XML fragment with the result of the validation.
- `/TGT` – It is created when parsing successfully the `/login` URI. It is the base for accessing resources using CAS. It is stored in a cookie, and is referenced every time the user has to be logged in transparently.
- `/TGC` – The Ticket Granting Cookie is set by CAS when initiating an SSO session, and is used for authenticating the user. It is automatically destroyed upon closing the browser, or parsing the `/logout` URI.
- `/LT` – The Login Ticket consists of a stream of characters generated by the parsing of the `/login` URI. This has the role of stopping repeated logins due to browser errors, or brute-force attacks.

All these tickets forcefully contain the full alphanumerical character set: a-z + A-Z + 0-9. Moreover, have to be of sufficient length, so it is impossible for
an attacker to anticipate them in due time.

CAS is based on the Kerberos model. It uses the TGT, in which it stores the session’s ID. This is the only cookie that returns to the server. This cookie allows the server to permit the user’s access to a resource without challenging him for credentials. An application can see only its own ST, associated to the users TGT in the server. This ST is an unique value, and it is invalidated when the application validates it for the first time. It is not used as a session key, so the implementation has to set its own state management.

CAS has to be deployed in a valid J2EE environment, like the demo application uses Tomcat. It can also use load balancing and clustering. The demo application was built with security in mind, so, the user interface, the core servlets and the ticket cache, are all separate. This cache contains the login status to the additional login server, in the case of this demo application an LDAP is used.

In the next figures three cases are presented: the first one, fig 5, a user tries to access a protected resource without being authenticated. In the second one, fig 6, the user first authenticates to the server, after which he accesses the desired resource. In addition, in the third one, fig 7, the user accesses the resource after being logged in from a previous session.

Over the years, CAS was found to have some minor vulnerabilities, Secunia.com, filled 6 reports for the PHP implementation of this protocol. (8) With these in mind, the demo application has a few improvements. All the cookies are sent over HTTPS. The credentials that are stored in the cache do not contain any personal data. They contain only session keys.

Cached credentials are not susceptible to replay attacks, the service keys only being sent to the server hosting the application, while the authentication keys are only sent back to the CAS server; this is why no service should run on the same server as the CAS authentication service. The user has the single logout function implemented.

Implementation of the unique use STs. Applications can deny SSO and force the CAS server to re-authenticate the user.

The application can also work without cookies, but the user will be asked every time for credentials. The core cache is stored in RAM, and cleared on every restart of the service. Furthermore, for the demo application, for the resources protected by HTTPS, the application requests the server certificate, for eliminating the impersonation and Man-in-the-Middle attacks. In addition, a list of permitted protocols, domains, hosts and applications was set up. These were implemented for stopping DoS attacks. Before serving requests, the server firstly checks this list.
3. Implementing a secure enterprise SSO solution

The whole demo application’s architecture is based on CAS server 3.4.6, deployed in a Tomcat 7.0.5 container. Furthermore, every application is protected through a personalized CAS client module and custom configuration files for the entire platform.

After configuring the Tomcat container to accept only HTTPS connections, the CAS server’s certificate was added in the JVM trust store for validation. For refined security, and distinguishing from regular users, power users or administrators, an LDAP has been implemented. Again, for improving security and assuring DoS protection, as it was said in section 2.2, a list of allowed applications has been set up. This list is checked before processing any requests.

In figure 8, the structure of the CAS setup is described. Two Tomcat Servlets running on a different server and a Zimbra Collaboration Server running in a CentOS environment, all of which are protected by the CAS server.

3.1. Authentication flow for the protected resources with CAS

After successful authentication to the CAS login page and LDAP filtering, a TGT is created, similar to the one bellow.

Based on this TGT a unique use ST is created. Moreover, it is uniquely associated to the requested resource by the following URI.

The ST is validated by parsing the URL as follows:

When the user tries to access the second protected servlet, after being authenticated, another unique ST is created based on the user’s TGT for the requested resource, like bellow.

After the former step, completes successfully, the ST can be verified and the user can be allowed access to the requested resource.

Another important ability for a SSO platform is the Single Log Out functionality, implemented in the current solution as a link in the portal. Upon pressing this button the user’s TGT is destroyed and he is redirected to a page notifying him of this fact.
In addition, the Zimbra server and the PHP portal also, have SLO functionality.

3.2. OpenID implementation

The desired functionality was introduced using myOpenID as an identity provider and a personalized version of the PHP class SimpleOpenID for interfacing the identity provider with the relying parties.

When presented a login screen, all the user has to do is provide his authentication string, an URI like http://IonitaM.myopenid.com/. Upon receiving it, the authentication string is parsed and sanitized, with regard to the second section of this article, such that, particles like "xri://" are removed, if present, and particles like "https://" are added if missing. If the first character of the authentication string is a "=" or "@" or "+" or "$" or a "!", then the string is parsed as an extensible resource identifier. Following the normalization phase is a discovery one, in which the relying party establishes communication based on the information based on parsing the provided URI.

In the portal of this demo application, anyone with a valid OpenID can view the secured resource. On the virtual shop called “AnuntA” written in PHP, the administration page has restricted access to only those that poses level 1 in a MySQL database. The passwords in this database are stored using SHA-512. In addition, the shopping cart section is available only to the signed in customers, those who possess level 2 in the database.

The user is redirected to the IdP’s login page by the following link.

https://www.myopenid.com/submit_password?u=079b-6462-8b08&sid=80d57176f0e7ebe-79319e6078f5807b178b6c00000000009c44

The application that request access to a protected resource on behalf of the user is verified by a link like the following one.

https://www.myopenid.com/submit_login?u=52a307e9-84d2-8808&sid=65580f8b&txkey=26f490f6d4a%20714a087725a%2046090000000002H4
If the two steps are successful, the user is authenticated and redirected to the desired resource.

3.3. NTLM implementation

Though less secure than the protocols referred above, NT Lan Manager is still used in some organizations. The second version of Microsoft’s protocol is more secure after patching it, based on the observations of Amplia Security in February 2010 (9). The weaknesses are present due to cryptographic weaknesses that permitted guessing the nonces, furthermore allowing a credential forwarding attack.

This protocol is still present in the newest versions of Microsoft’s operating systems for backward compatibility reasons, but is slowly upgraded to the Kerberos protocol, which offers similar benefits, with a higher degree of security.

NTLM was implemented in the Multi-SSO for allowing older applications to integrate with the new environment. It’s using the second version of the protocol, through the NTLM PHP library. It uses two 16-byte answers from the client, when an 8-byte request is received from the server. The answer is made up from the HMAC-MD5 hash of the request, a random generated challenge from the client side and the HMAC-MD5 of the user’s password. The shorter answer is organized as an 8-byte randomly generated number. To be able to verify the response, the server has to receive the client’s challenge. For this short answer the 8-byte challenge is added to the 16-byte response, forming an 24-byte package, compatible with the 24-byte answers used in the first version of the protocol. The second answer used in NTLMv2 includes the current time and date in NT format, a random 8-byte value, the domain’s name. Therefore, this second response is variable in length. (10)

Upon researching the integration of this protocol, it was found that only Internet Explorer gives full compatibility with the protocol. In the protocol description (11), it is stated that the NT passwords are stored as MD4 hashes of the UTF16LE strings. After obtaining these strings the verification process can begin. Getting these strings in Windows 7 SP1 can be a difficult task, so, for this demo the ntlm_md4(ntlm_utf8_to_utf16e(array)) function from the PHP-NTLM binary include was used to create an MD4 hash of an array containing the user name and his password. For authenticity verification the pair username (MD4 hash of the password) is used, for example: Mihai(****97BA0A691****322122DF1D D****).

Figure 12. Password challenge for accessing the protected resource

4. Conclusions and future research

An important aspect regarding SSO is the centralized platform that has to protect the components that gave up on their own authentication methods for the added security and comfort of SSO. The reliability of the demo setup was evaluated using JMeter. This certified that the demo setup could serve 800 logins per second from a valid source. After which the test setup becomes sluggish. Increasing the number of concurrent connections makes logins to fail because of the 30s timeout.

The present demo architecture was designed as a Secure-SSO scheme, taking into account the improvements brought to it, over the default configuration. With little modification, this setup can easily function in a production environment.

The only downside of the developed solution is the prerequisite for a JavaEE
environment. A commercial one being expensive, but of course, for keeping the total cost of ownership close to zero, an open source environment like the presented Tomcat can be used. It has to be mentioned that setting up and maintaining such a platform in a production environment is technically challenging, and requires experienced personnel.

In the future, an important feature for improving the solution security wise would be implementing fully DoS resistant systems by introducing Threshold Puzzles, for the client. This would make the login effort greater for the client than the server, discouraging DoS attacks. (9) Furthermore, by using a multi-CAS server setup, a load balancer can distribute the load between servers, even in larger setups, with even more logins per second.

In addition, measures that are more active can be taken like moving the SSO server, physically. The server can change its IP address and announce it only to the trustworthy clients on the network by means of an XML message, the attacker being blacklisted will not receive these messages. (10)

Acknowledgement

Parts of this research have been published in the Proceedings of the 5th International Conference on Security for Information Technology and Communications, 2012

References


