Web Access Management and Single Sign-On

Ronnie Dale Huggins

In the "old days" of computing, a user would sit down at his or her workstation, login to the desktop, login to their email system, perhaps pull up a thick client or two, login to those as well, and maybe access a couple of internal web pages, and again, login to those. Now take into account that all of these systems are likely handled by different support groups, so the user has to not only remember user IDs for all of these applications, they need to remember the password associated with these systems. All of these security silos eventually become an administrative nightmare and result in a decreased user experience. Costs associated with helpdesk calls for password resets also increase.

With the proliferation of online, or web-based access, users are nowadays using less thick clients and more web-based applications. While this has helped to alleviate some of the problems associated with managing access to thick clients, the application security silo challenge still exists. Some of this was addressed by implementing strategic directories such as LDAP or Active Directory (AD), where the developers could code LDAP lookups to challenge users for credentials that were in a central location. This eliminated the need for multiple user IDs and passwords on web based systems. Pure Microsoft shops could also benefit by using domain authentication to sign users into their desktops, and use IIS as the web server platform that would sign the users in with Integrated Windows Authentication (IWA). While this seems like the way to move forward with web based access, other challenges still linger. What is a company uses AD but isn't a pure IIS shop? Suppose half of the web platform is Apache, or iPlanet based.

What now? How do companies drive adoption of new web applications without the need to burden its user base with addition credentials?

The answer lies in technology known as Web Access Management (WAM), also commonly known as Single Sign-On (SSO). There are many commercial and open source products available such as CA's Siteminder, IBM's Tivoli Access Manager, and Sun's OpenSSO. There are many others, these just serve as an example. So how does WAM technology ease the burden to the users, while allowing secure access to sensitive data? Each of these solutions uses browser-based cookies to control access to resources that are being accessed. (topbits.com) While it sounds simple enough, it's a little more involved that just issuing a browser cookie and sending the user on their merry way. Consider the following:

- A company wants to bring a new HR application online
- Access needs to be controlled so that only employees (e.g., no contractors) can get in
- Access needs to be centrally managed

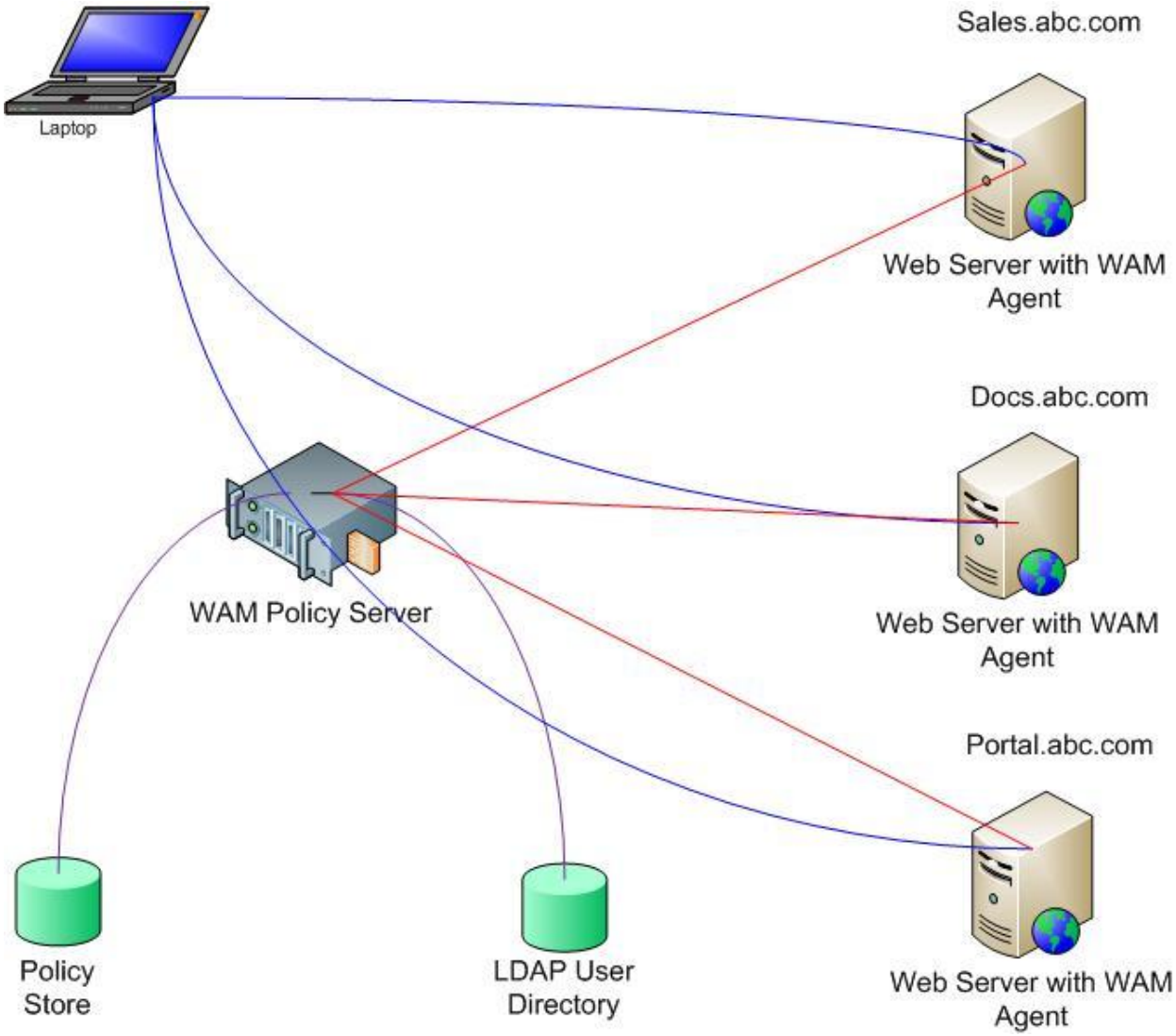
How would a WAM solution help in these use cases? In a typical WAM deployment, there are a few components (these are not all of the components):

- A policy server
- A policy store
- Web agents that reside on the web server to control access

The WAM solution will leverage the same user directory discussed earlier, so the user would be authenticating with the same credentials they are already familiar with. The administrator of the web server where the application is hosted would install the WAM agent on the server, and

register that agent with the WAM policy server, set up and apply policies, and bring it online. Now granted, this is a high level example but is intended to paint the picture of what the solution does.

Behind the scenes, a lot is going on. The following diagram shows what a simple WAM solution might look like:



The blue lines represent traffic between the user's browser and the resource they are trying to access. Red lines represent the WAM agent intercepting the request, and asking the policy

server whether or not the resource trying to be access is protected, whether the user is authenticated, and whether the user is authorized to access that resource. Purple lines indicate the policy server performing lookups against its policy store and lookup up the presented credentials in the user directory. When the user attempts to access a protected resource, the agent intercepts the request, and asks the policy server if it's a protected resource. If the policy server returns "no", then the user is presented with the requested page. If it returns "yes", the agent then asks if the user is authenticated. If the policy server returns "no", then the user is challenged for credentials. If the policy server had returned "yes", the agent then would ask if the user was **authorized** to access that resource. If the policy server returns "yes", then the user's browser is issued a browser cookie and is granted access. That same browser cookie is passed in the HTTP header and is used to authenticate your identity to any other resources protected by the WAM infrastructure. (Twist, 2005) The cookie will stay active until either the browser is closed, or an arbitrary timeout is reached as set by the administrator.

This type of scenario works great in an enterprise where all resources reside in the same cookie domain, such as abc.com. Once the user is authenticated and authorized to access docs.abc.com, the user is free to browse to other resources protected by the WAM infrastructure such as portal.abc.com or sales.abc.com, provided that the user is authorized to access those resources. Now assume company abc.com has another internal domain such as def.com, and the def.com domain is protected by the same WAM infrastructure. What would happen if the user, Jane, has been authenticated and authorized into docs.abc.com, and she browses over to hr.def.com? Even if the administrator had a policy in place for hr.def.com that allowed Jane access to the web application, Jane would be challenged again for credentials. Her browser cookie is not valid in the def.com domain.

The reason for this, cookies are inherently limited to the scope of the originally issued cookie domain, abc.com. (Whalen, 2002) There are two ways to go about solving this problem, and depending on where the resource is logically hosted will determine which method is better suited at solving the problem. The end result is the same, extending SSO beyond the scope of the original cookie domain.

The first method is deploying a cross domain cookie provider. (Pouttu-Clarke, 2005) The cookie provider, in this scenario, would be a WAM agent specially configured to issue cookies valid in another domain. Upon request, the cookie provider will be contacted by the agent in the def.com domain, read the original cookie from abc.com domain, and issue a second browser cookie valid for the def.com domain. So rather than being challenged for credentials again, when Jane accessed hr.def.com, she would be granted access based on the cookie provider model. This method is more cost effective and easy to deploy when used internally. If the resource is hosted externally, then the second solution should be examined.

Let's assume that our fictitious ABC company uses a 3rd party for 401k, and also subscribes to vendor supplied Software as a Service (SaaS) offerings such as Google Apps or SalesForce.com. ABC wants to extend SSO to these service providers that are obviously hosted outside the scope of our abc.com domain. SSO can be extended by leveraging identity federation and the Security Assertion Markup Language, also known as SAML. A full-blown SAML discussion is beyond the scope of this document, however I will go into some detail. SAML is an OASIS standard whose goal was to provide an XML framework to allow authentication and authorization from an SSO perspective. (Lewis & Lewis, 2009)* When incorporating SAML into the WAM framework, SSO is being leveraged as a web service that is transparent to the user. The user, Jane, is authenticated in the abc.com domain, however the target may be in the salesforce.com

domain. In this scenario, the browser cookie is used to generate a SAML token that is passed by a web service over to the salesforce.com domain, where the SAML token is decrypted, and the user is authorized by salesforce.com. This is a typical identity provider/service provider model, where ABC company is providing the identity and salesforce.com is providing the service. Since SAML is a standard, ABC company might be using Siteminder to generate the SAML token while salesforce.com might be using Shibboleth as the consuming party, and both can exchange security information without interoperability problems. (Lewis & Lewis, 2009)* This is a sample SAML assertion to show what data being passed between the two entities may look like:

<samlp:Response xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion" xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol" Consent="urn:oasis:names:tc:SAML:2.0:consent:unspecified" Destination="https://mypartner.com/metaAlias/sp" ID="ad58514ea9365e51c382218fea" IssueInstant="2009-04-22T12:33:36Z" Version="2.0"> <saml:Issuer>http://login.mycompany.com/mypartner</saml:Issuer> <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#"> SIGNATURE VALUE, ALGORITHM, ETC. </ds:Signature> <samlp:Status> <samlp:StatusCode Value="urn:oasis:names:tc:SAML:2.0:status:Success"> </samlp:StatusCode> </samlp:Status> <saml:Assertion ID="1234" IssueInstant="2009-04-22T12:33:36Z" Version="2.0"> <saml:Issuer>http://login.mycompany.com/mypartner</saml:Issuer>

<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#"> SIGNATURE VALUE, ALGORITHM, ETC. </ds:Signature> <saml:Subject> <saml:NameID>NAMEID FORMAT, INFO, ETC</saml:NameID> <saml:SubjectConfirmation Method="urn:oasis:names:tc:SAML:2.0:cm:bearer"> <saml:SubjectConfirmationData

NotOnOrAfter="2009-04-22T12:43:36Z" Recipient="https://mypartner.com/metaAlias/sp">
</saml:SubjectConfirmationData> </saml:SubjectConfirmation> </saml:Subject>
<saml:Conditions NotBefore="2009-04-22T12:28:36Z" NotOnOrAfter="2009-04-
22T12:33:36Z"> <saml:AudienceRestriction>
<saml:Audience>mypartner.com:saml2.0</saml:Audience> </saml:AudienceRestriction>
</saml:Conditions> <saml:AuthnStatement AuthnInstant="2009-04-22T12:33:20Z"
SessionIndex="ccda16bc322adf4f74d556bd"> <saml:SubjectLocality
Address="192.168.0.189" DNSName="myserver.mycompany.com">
</saml:SubjectLocality> </saml:AuthnStatement> <saml:AttributeStatement
xmlns:xs=SCHEMA INFO> <saml:Attribute FriendlyName="clientId" Name="clientId"
NameFormat="urn:oasis:names:tc:SAML:2.0: attrname-format:basic">
<saml:AttributeValue>1234</saml:AttributeValue> </saml:Attribute> <saml:Attribute
FriendlyName="uid" Name="uid" NameFormat="urn:oasis:names:tc:SAML:2.0: attrname-
format:basic"> <saml:AttributeValue>the.user@mycompany.com
</saml:AttributeValue> </saml:Attribute> </saml:AttributeStatement>
</saml:Assertion> </samlp:Response>

Figure 1: Sample SAML Assertion (Lewis & Lewis, 2009)*

As you can see, there are a lot of attributes passed along in the SAML assertion. There are name-value pairs that indicate the time that the assertion is valid, and so on. If the service provider side needs to know certain attributes such as email address, user ID, or any other data, they can be pulled from the identity provider's user directory and added to the SAML assertion to be used by the service provider.

The following is the SAML assertion element:

saml:AssertionType
Assertion
saml:Issuer
ds:Signature
saml:Subject
saml:Conditions
saml:Advice
0..□
saml:Statement
saml:AuthnStatement
saml:AuthzDecisionStatement
saml:AttributeStatement

These elements make up the components of the SAML assertion that are generated at the identity provider, and are passed to the service provider. Only the Issuer element is required. (Nordbotten, 2009)

To recap, it's been demonstrated here how a WAM solution can ease the administrative burden of managing security silos, reduce the need for developers to write authentication code into their applications, and how a WAM solution can be leveraged to extend the SSO user experience outside of the company.

Works Cited

*Lewis, K. D., & Lewis, J. E. (2009). Web Single Sign-On Authentication using SAML. *IJCSI International Journal of Computer Science Issues, Vol. 2, 2009* , 41-48.

*Nordbotten, N. A. (2009). XML and Web Services Security Standards. *IEEE COMMUNICATIONS SURVEYS & TUTORIALS, VOL. 11, NO. 3, THIRD QUARTER 2009* , 17-18.

Pouttu-Clarke, M. (2005, January 19). *Cross Domain Cookie Provider*. Retrieved April 9, 2010, from theserverside.com: http://www.theserverside.com/patterns/thread.tss?thread_id=31258

topbits.com. (n.d.). *Browser Cookie*. Retrieved April 9, 2010, from topbits.com: http://www.topbits.com/browser-cookie.html

Twist, J. (2005, November 16). *Better understand cookies with ieHTTPHeaders*. Retrieved April 9, 2010, from http://www.thejoyofcode.com/Better_understand_Cookies_with_ieHTTPHeaders.aspx

Whalen, D. (2002). *The Unofficial Cookie FAQ*. Retrieved April 9, 2010, from cookiecentral.com: http://www.cookiecentral.com/faq/#3.3