WHITE PAPER | FEBRUARY 2015

Designing a CA Single Sign-On Architecture for Enhanced Security

Using existing settings for a higher-security architecture

ca technologies

Table of Contents

Executive Summary

Challenge

CA Single Sign-On (CA SSO) is widely deployed across the globe to secure and provide single sign-on for a wide range of applications with different security needs. CA SSO uses multiple methods for maintaining user sessions—the most commonly deployed option is to use cookies. In many cases, administrators will configure CA SSO to send these cookies to a wide variety of Web servers—including those that do not need access to the session cookie. By configuring any application to send a cookie to a wide scope of servers, one creates a vulnerability in the architecture that allows an attacker to steal cookies and replay stolen session cookies to impersonate an authenticated user.

Opportunity

CA SSO utilizes a patent pending "Enhanced session assurance with Device DNA™" approach, to help mitigate the risk of session replay attacks. CA SSO offers several different settings that allow session security to be enhanced, including the use of "host only" cookies—session cookies that are designed with the intent to only be transmitted back to the host that created them. This approach can be used while delivering broader, cross-application SSO for end users while allowing agents to communicate from one domain to another using a central cookie provider. The cookie provider can provide a one-time use reference to a session stored in a centralized session store to pass the session from one application to the next.

Benefits

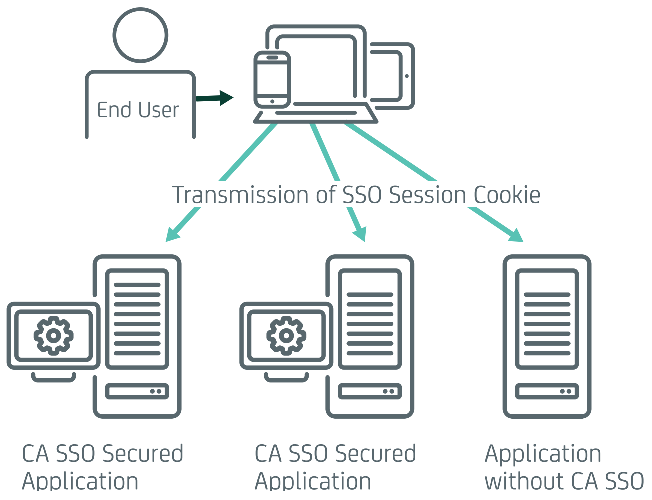
CA SSO offers administrators the ability to configure how the architecture should behave for all or a subset of their applications. Use of a "host only" architecture can enhance security by making it much harder for an attacker to capture a session cookie while limiting the exposure of a stolen session to a specific application, until such time as the idle timeouts or the session assurance feature can detect and stop the stolen session.

Section 1:

The Importance of Securing CA SSO Sessions

Session hijacking, also known as cookie hijacking, is not a new threat. It has evolved into an almost permanent security risk since HTTP 1.1 became a standard and is not limited to CA SSO session tokens. The OWASP Foundation has identified that session stealing is part of attack A2 “Broken Authentication and Session Management” on its 10 Most Critical Web Application Security risks. If the session is stolen by an attacker, it can be replayed, and Web applications will show the information in the context of the stolen identity to an attacker. Additionally, all logs will record the attacker’s requests as coming from a valid authenticated user, making the attack extremely hard to detect.

In most deployments, the CA SSO session is configured to be shared across all Web applications sharing the same cookie (DNS) domain (e.g. any Web server in ca.com). This is the easiest way to help ensure the cookie gets to all of the intended CA SSO applications. It is also the most vulnerable, since the Web browser will provide the CA SSO session to applications that need it and to applications that do not and all applications share a common session token.

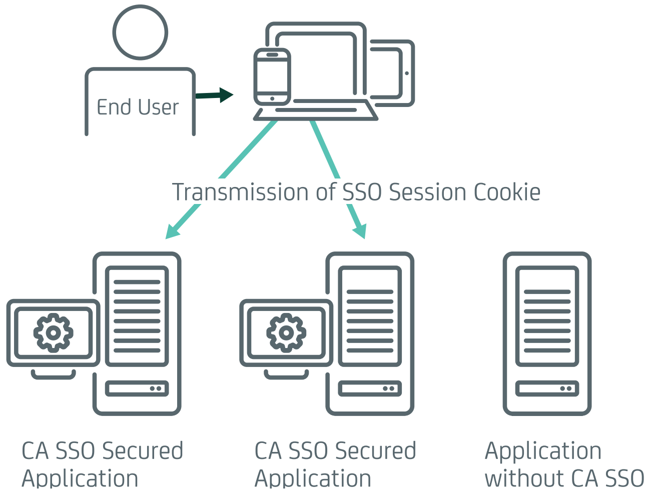

End User
Transmission of SSO Session Cookie
CA SSO Secured Application
CA SSO Secured Application
Application without CA SSO

Section 2:

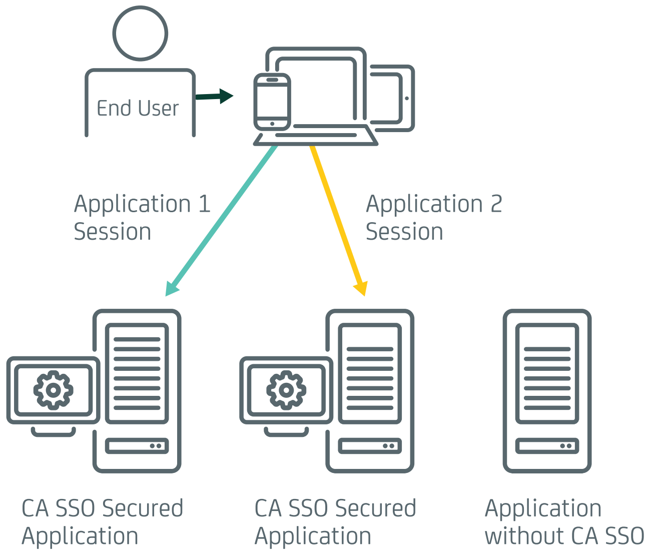
Key Settings to Change the CA SSO Behavior

There are several settings that can be used to increase the level of security to prevent a session from being inadvertently passed to applications that do not need it. These settings will alter the typical CA SSO behavior to adapt to more secure scenarios.

The first behavior change is to only send the CA SSO cookie to applications that require it. This prevents passing the cookie to sites that have no reason to view the session. This is done by changing the scope of the cookie distribution to specific hosts instead of all servers in a common domain.


End User
Transmission of SSO Session Cookie
CA SSO Secured Application
CA SSO Secured Application
Application without CA SSO

CA SSO can also be configured to use specific sessions for specific applications—further preventing the risk that if a session has been stolen that it can be used for multiple applications.


End User
Application 1 Session
Application 2 Session
CA SSO Secured Application
CA SSO Secured Application
Application without CA SSO

Key Settings to Control CA SSO Session Security

The following settings have been in CA SSO for several years and are configuration changes to the behavior of agents and gateways. The following settings can all be found in the Agent Configuration Object (ACO).

CookieDomain

The CookieDomain setting is used to define the cookie domain value used to create cookies using the set-cookie HTTP response header. This setting defaults to empty string, indicating to the agent that the cookie domain should be derived from a request's HTTP_HOST header based on the CookieDomainScope setting described below. A value of "NONE" indicates that no cookie domain value should be set. This defines a "server-only" cookie. Alternatively, a specific cookie domain value can be set (e.g. ".app.ca.com"). Setting a specific cookie domain value should be used with caution since the domain specified for a cookie must match some part of the domain of the request over which it is issued. As a result, the cookie domain value cannot be arbitrary. If a given Web agent will service requests coming to multiple HTTP hosts, then a specific cookie domain value should NOT be used and in fact is seldom necessary.

CookieDomainScope

The CookieDomainScope setting controls the scope of a session by defining how a cookie domain value will be derived from a request's HTTP_HOST header. The default value is 0. This value indicates global scope, which defines the cookie domain at the top domain (e.g. "ca.com"). A value of '1' is not allowed, since ".com" and ".net," etc. are not legal cookie domains. '2' is identical to stating '0'. Values larger than 2 indicate a more defined scope where the domain of the HTTP_HOST allows it. For example, a value of '0' or '2' would result in the cookie domain ".ca.com" being set for HTTP_HOST "myserver.security.ca.com." A value of '1' is not allowed (ignored in favor of default '0' value), and a value of '3' would result in a cookie domain of ".security.ca.com" being set. A value of '4' would define a value of "myserver.security.ca.com." However, in that case it is more appropriate to set CookieDomain to "NONE" as described above. CookieDomainScope is ignored when CookieDomain has been set to NONE, indicating that "server-only" cookies should be used. In the case of server only cookies, the scope is ALWAYs the full value of HTTP_HOST minus any provided port value.

CookieProvider

Because we will be using host only cookies, and still want to have SSO between applications, there must be a centralized identity provider site that can provide the session information to other applications. This is the CA SSO CookieProvider. It is a centralized server that is designed to pass session information to other remote Web applications. Any CA SSO gateway or agent can serve as the cookie provider. Agents that use this CookieProvider use the CookieProvider URL specified in the ACO setting.

EnableCookieProvider

The EnableCookieProvider tells a SSO gateway or agent that it can function as a cookie provider. It is recommended to turn this setting off on all agent ACOs except for the intended cookie provider. This can prevent an attacker from escalating privileges if they have stolen a CA SSO session for one application and attempt to use it for getting into other applications.

StoreSessionInServer

Traditionally CA SSO cookie providers have placed the session into the HTTP query string data as part of a redirect to the final target. Placing this data in the query string may allow attackers to gain access to the session. Instead of that approach, one can tell the CA SSO cookie provider to store the session in a centralized session store and then pass a one-time reference to the stored session on the query string. Then the application requesting the session will have the session provided by the policy server it is communicating with instead of directly reading it from the query string. This approach is much like SAML artifact profile.

LimitCookieProvider

When using a centralized cookie provider, the cookie provider can be used to create new CA SSO session cookies for remote agents, or allow a remote agent to create a new session at the cookie provider if the user directly logged into the remote site. This setting can force all authentications to occur within the domain of the central cookie provider and will reject sessions if they are created at remote applications. Use of this setting will depend on security and business policy. If you can use a centralized location for all login pages, it is recommended to use this setting.

TrackSessionDomain

To help ensure that a CA SSO session is only used for the site it was intended, the TrackSessionDomain ACO setting can be used. This setting instructs the Web Agent to encrypt and store the intended domain of a session within the session cookie itself. During subsequent requests, the Web Agent compares the intended domain stored within the session cookie against the domain of the requested resource. If the domains do not match, the Web Agent rejects the session cookie.

TrackCPSessionDomain

A CA SSO cookie provider is responsible for handling the transformation of a CA SSO session from one domain to another. To allow this transformation to work correctly when using the TrackSessionDomain, the cookie provider must be instructed to rename the domain inside of the session so it can be used at other locations. The TrackCPSessionDomain setting tells the cookie provider to validate the domain of the cookie it has prior to transforming it for another application. This prevents an attacker from using the cookie provider to transform cookies arbitrarily from one domain to another (e.g. sending the cookie provider a stolen ".app1.ca.com" cookie to be transformed to ".app2.ca.com").

ValidTargetDomain

The ValidTargetDomain parameter identifies the valid domains and hosts for remote systems during processing. Before the user is redirected, the agent compares the values in the redirect URL against the domains in this parameter. Without this parameter, the agent redirects the user to targets in any domain. This setting is used to avoid cross-site attacks through redirects to the login pages, cookie providers and session assurance URLs.

Section 3:

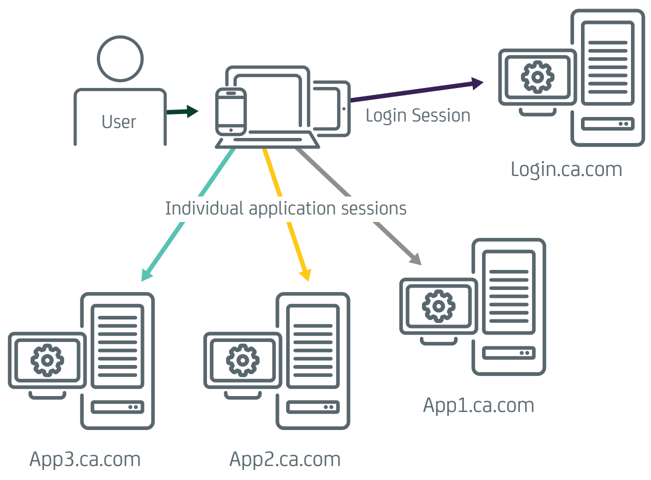
Designing an Architecture for Maximum Security

Use of these settings can allow for an architecture that forces each application to have a separate session that can only be used for that application, requiring all users to be authenticated at a central location, while maintaining SSO functionality.

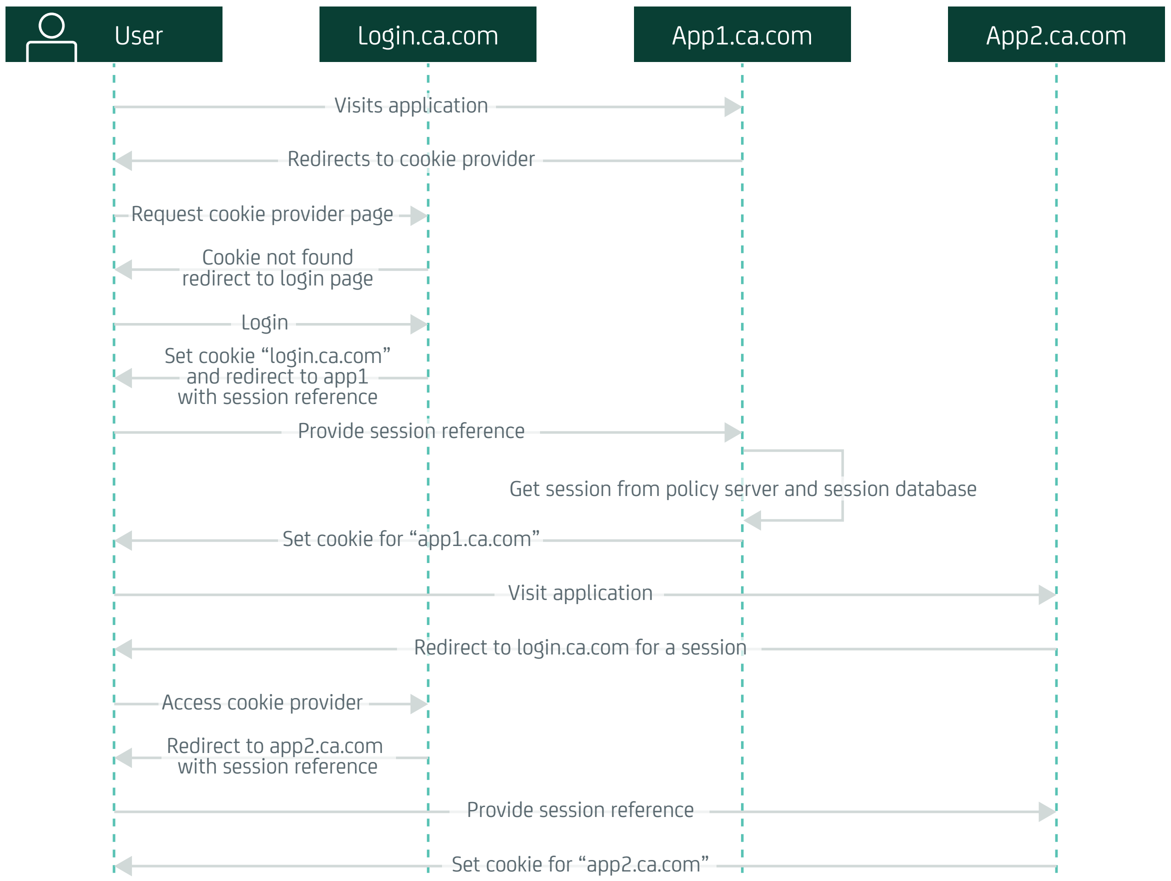
In this example, we have one central site, login.ca.com, which serves as our cookie provider and hosts the login pages and multiple applications.

	Default Setting	Login.ca.com	App1.ca.com	App2.ca.com	App3.ca.com
CookieDomain	"" (empty string)	NONE	NONE	NONE	NONE
CookieDomainScope	0 (use top domain scope)	Default	Default	Default	Default
CookieProvider		Default	https://login.ca.com/siteminderagent/SmMakeCookie.ccc		
EnableCookieProvider	Yes	Yes	No	No	No
StoreSessionInServer	No	Yes	Yes	Yes	Yes
LimitCookieProvider	No	Yes	No	No	No
TrackSessionDomain	No	Yes	Yes	Yes	Yes
TrackCPSessionDomain	No	Yes	Default	Default	Default
ValidTargetDomain	All domains ("")	App1.ca.com App2.ca.com App3.ca.com	Default	Default	Default

The above settings result in an architecture that looks like this:



A high-level view can be seen below:

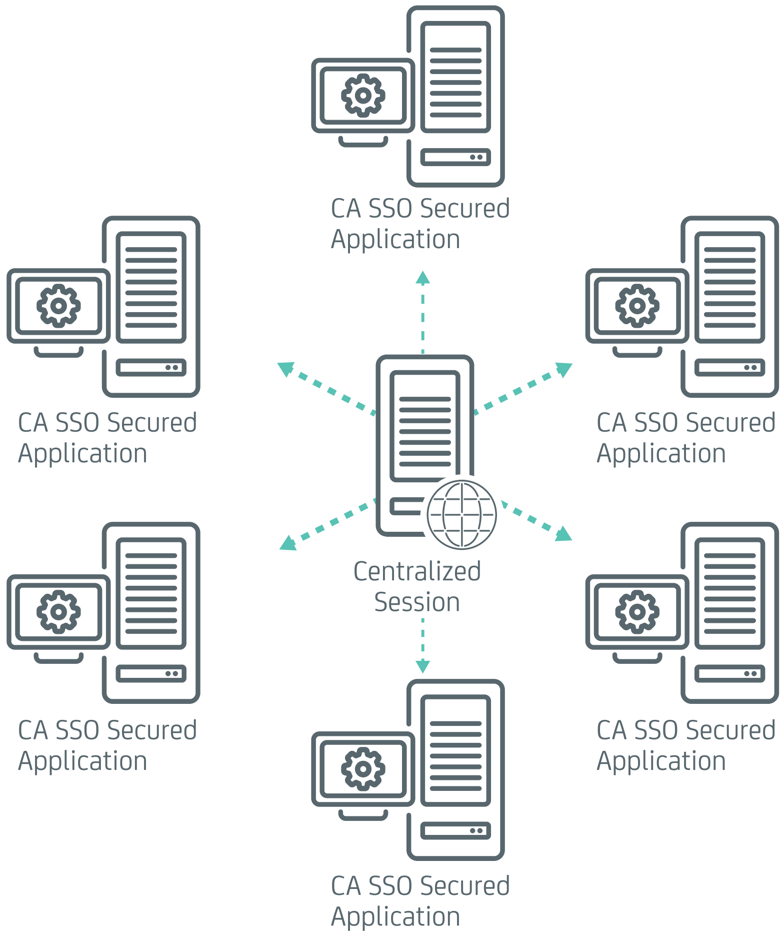

User
Login.ca.com
App1.ca.com
App2.ca.com
Visits application
Redirects to cookie provider
Request cookie provider page
Cookie not found redirect to login page
Login
Set cookie "login.ca.com" and redirect to app1 with session reference
Provide session reference
Get session from policy server and session database
Set cookie for "app1.ca.com"
Visit application
Redirect to login.ca.com for a session
Access cookie provider
Redirect to app2.ca.com with session reference
Provide session reference
Set cookie for "app2.ca.com"

This architecture, combined with other controls for CA SSO sessions including SSL Only and HTTP Only cookies, Enhanced Session Assurance with DeviceDNA for device fingerprinting and proper Idle/Session timeout policies can lock down a CA SSO environment to enhance session security while still maintaining SSO for end users.

Section 4:

Conclusions

As organizations deploy advanced risk-based and multi-factor authentication systems, security of the session tokens provided after authentication is becoming more important, as they are the next logical vulnerability in the infrastructure. CA SSO can perform single sign-on and session management across a wide variety of sites using host-only cookies and can use device fingerprinting to validate the session is coming from the host it was issued for. The use of an architecture that uses host-only cookies makes the theft of session cookies much more difficult by using a centralized star topology instead of a single session for the entire domain.


CA SSO Secured Application
CA SSO Secured Application
CA SSO Secured Application
Centralized Session
CA SSO Secured Application
CA SSO Secured Application
CA SSO Secured Application

This architecture also limits the scope of any exposure should a session become compromised. Since each application has a separate session cookie, should a session get stolen, the stolen session cookie can only be used for the application which it was intended for and cannot be used to gain access to other applications until the session assurance, timeouts or other controls can invalidate the stolen session.

Section 5:

References

OSAWP top 10 list: https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

Section 6:

About the Author

Aaron Berman is currently a Senior Advisor for the CA Technologies field organization, with responsibilities that include product and sales strategy, messaging and working as an extension to product management. He has over 15 years' experience troubleshooting, designing, implementing and defining strategy for Web access management solutions, including architecting CA Single Sign-On (formerly CA SiteMinder) and CA Identity Manager (formerly CA IdentityMinder) 100 Million user load tests, and leading multiple Federation Interop events. Prior to joining CA Technologies and his work with Netegrity, Aaron managed support and presales services for Web access management solutions for Raptor Systems/Axent Technology. Aaron was previously a VP, Principal Architect for CA Technologies services organization. Aaron has a Bachelor's of Science in Computer Science from Syracuse University.

For more information, visit **ca.com/secure-sso**

1 Lhttps://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

CS200-112938_0215