

Netcontinuum Single Sign On PRD

<i>Revision Number</i>	<i>Date</i>	<i>Author</i>	<i>Changes</i>
1	04/23/2007	sraghavan	Initial draft
2	04/27/2007	sraghavan	Updated after first round of comments
3	05/09/2007	sraghavan	Siteminder changes
4	05/16/2007	sraghavan	Updated logs and client ip checks
5	05/22/2007	sraghavan	Updated with review comments

Table of Contents

1. Overview.....	3
1.1 Purpose.....	3
1.2 References.....	3
1.3 Terminology.....	3
2. NC SSO Overview.....	3
2.1 Current Design Limitations.....	3
2.2 Netcontinuum Single Sign On (NC SSO).....	4
3. NC SSO Implementation.....	4
3.1 Configuration.....	4
3.2 NC SSO - Stateless AAA.....	5
3.3 Cookie Encryption.....	6
3.4 NC SSO Checks.....	6
3.4.1 Authentication database validation.....	6
3.4.2 Creation and Idle timeouts.....	6
3.4.3 Client IP checks.....	6
3.4.4 Authentication level checks.....	6
3.4.5 Access Control policies checks.....	7
3.5 Multi Domain SSO.....	7
3.5.1 Slave domain first.....	8
3.5.2 Master domain first.....	8
3.5.3 Logout.....	9
3.5.3.1 Inform the user.....	9
3.5.3.2 Chained Logout.....	9
3.5.4 NCSSOTOK acceptance.....	10
3.5.5 Idle timeouts.....	10
4. Open Issues.....	10
4.1 Cookie Timeouts.....	10
4.2 POST Redirects.....	10
5. Impact on current AAA behaviour.....	10
5.1 SSO by default.....	10
5.2 Support for multiple sessions.....	11
5.3 Stateful model dropped.....	11
5.3.1 Basic Authorization header insert.....	11
5.3.2 Rewrite macros.....	11
5.4 State Failover.....	11
6. Requirements and Assumptions.....	12

6.1 GMT Timestamps.	.12
6.2 Time synchronization.	.12
7. Impact on modules.....	.12
7.1 UI Validations.	.12
7.2 Statistics.	.12
7.3 New UI commands.	.12
7.4 Logs.	.12
7.4.1 System logs.	.12
7.4.2 Web-Firewall logs.	.12
7.5 SNMP.	.13
7.6 Upgrade.	.13
7.7 Development Tools.	.13
7.8 Documentation.	.13
8. Future Enhancements.....	.13
9. Implementation Notes.	.13
9.1 RPC APIs.	.13
9.2 RPC Data Structures.	.13
9.3 SAPD commands.	.14
9.4 File Formats.	.14
10. Test Plan.	.14
10.1 Dev Test.	.14
10.2 QA Test recommendations.	.15
11. Future Enhancements.....	.15
12. Alternatives considered.....	.15
13. Task List.....	.15

1. Overview

1.1 Purpose

This document describes the implementation and configuration changes for supporting Netcontinuum Single Sign On (NC SSO). NC SSO will address both single and multi domain scenarios. This document will also describe how NC SSO can be used as a solution for SFO of authentication sessions.

1.2 References

[MRD]: references to MRD, specific sections.

[CUST]: Any customer documents

[ENG]: Feature Request bug [26846](#).

1.3 Terminology

WAF: Web Application Firewall. Also refers to devices sold by NetContinuum.

AAA: Authentication, Authorization and Accounting, In this document it refers to the AAA solution offered by WAF.

Master URL: Master URL in a multi domain SSO setup. Same a cookie provider url in siteminder.

SSO: Single Sign On

NC SSO: Single Sign On solution provided by WAF.

Cache URL: The reduced url that will be used in the cache list (explained later). The reduced url will be derived from the actual request url using the configurable resource depth parameter under the access-control container (explained later). This entity will be used for both cache storage and match.

User Session Cookie: This refers to the cookie that is used to uniquely identify the issuer of the request. This will be the NCIS_USRSESSION_ID cookie. This document will provide details of the value of this cookie.

SSO User Session Cookie: This refers to the cookie that is used to uniquely identify the issuer of the request in an SSO setup. The cookie name is NCIS_USRSESSION_ID and this document will provide details of the value of this cookie.

SFO: Stateful Fail Over.

Roles: Refers to the list of roles/groups that a authenticated user belongs to. Roles and groups are used interchangeably in this document.

2. NC SSO Overview

2.1 Current Design Limitations

The current AAA implementation on the WAF is implemented using a stateful design. When a user performs a successful login, the WAF creates a session for the user in the box. This session is added to a lookup trie for efficient searching. The encrypted lookup key is sent back to the client as the user session cookie value. Upon receiving this cookie, the value is decrypted and used for searching the session in the trie. The user's session on the box contains information that will be used while applying the configured access control policies.

While this is a sufficient design it simply does not scale. A setup like for e.g. JPMC where multiple WAFs with identical configuration are used for better throughput, the current design will not work. The session cookie cannot be used across the WAFs since it is tied to an in memory session on the WAF that processed the successful login request.

The current AAA implementation does not support migration of AAA user sessions from one WAF to another in a cluster setup. As a result when a failover happens all authenticated users are forced to login again.

2.2 Netcontinuum Single Sign On (NC SSO)

NC SSO is proposed as the feature to address the two major limitations detailed in the previous section. A stateless session maintenance also provides the added benefit of reduced memory usage.

NC SSO will operate by maintaining the relevant parts of user session in the session cookie instead of an in memory structure. By relevant parts the reference is to those fields in the user session that are needed for applying access control policies. These fields are

- User name
- User Roles
- IP Address of the host from where the login was performed
- Creation and last access time

The simplest design will be to create an encrypted cookie value of all these fields for the NCIS_USRSESSION_ID cookie. This approach however has its limitations, consider two WAFs NCG-1 and NCG-2 with web-applications WAPP-1 and WAPP-2 respectively. Both these web-applications are bound to authentication databases AUTHDB-1 and AUTHDB-2 respectively and the two authentication databases are not identical. User “developer” performs a successful login on WAPP-1 and receives the user session cookie with value in the format as stated above. The user now moves on to access some urls in WAPP-2 and WAPP-2 decrypts the cookie value and applies the access control policies. This is a security hole for the following reasons

1. user “developer” need not exist in AUTHDB-2
2. user “developer” might be different from “developer” on AUTHDB-1, i.e a different password or a different set of roles.

Under these circumstances blindly accepting the cookie will not work. The alternatives are to validate the credentials also on AUTHDB-2 or not accept the session cookie if the two authentication databases are not identical. The former approach is not advisable since the password of the user must be added to the cookie value for credentials validation. Also, maintaining a single cookie for web-applications bound to different authentication databases comes with its own share of problems.

NC SSO will only be supported with identical authentication databases. How exactly we achieve this will be explained later when the format of the SSO User Session cookie is explained.

NC SSO will support both single and multi domain SSO. While single domain is pretty straightforward, multi domain SSO will be achieved by designating a Multi Domain Master for distributing the SSO User Session Cookie to all the remaining slaves. This is similar to the model used by CA Siteminder.

NC SSO is proposed for only ldap and radius authentication databases. The existing CA Siteminder and the proposed RSA integration solutions come with their own SSO support. NC SSO is not being proposed as an alternative to these SSO models.

WAF will work in SSO mode by default when authentication is turned on. Configuration parameters will be provided to extend this functionality to multi domain SSO. The old stateful method will not be supported anymore.

3. NC SSO Implementation

3.1 Configuration

```
[cluster]
  [boxname]
    [settings]
      user-session-timeout          --> remove, moved under [AAA]
      allow-multiple-user-sessions  --> remove
      authorization-cache-timeout   --> remove, moved under [AAA]
    [AAA]
      auth-user-session-idle-timeout --> New Container
      auth-user-session-idle-timeout --> idle timeout in minutes (Default: 15)
```

```

auth-user-session-creation-timeout --> creation timeout in minutes (Default: 60)
auth-user-session-update-period    --> cookie update in seconds (Default: 30)
authorization-result-cache-timeout --> cache timeout in minutes (Default: 30)
add-password-in-session-cookie     --> yes|no (Default: no)

```

The new AAA container will be placed parallel to the http container under cluster --> boxname --> settings.

```

[vsite]
[web-application]
[web-firewall]
[url-policies]
[url-policy]
[access-control]
    authentication-level = Possible values from 1 - 255 (Default: 1)
[authentication]
[siteminder-sso]
    status --> Remove parameter
[nc-ssso] --> New Container
    multi-domain-master = yes|no (Default no)
    multi-domain-master-url = <url> (multi domain processor)

```

3.2 NC SSO - Stateless AAA

The stateless cookie or the SSO User Session Cookie value will have the following format

<auth-db realm name>:<userid len>:<userid>:<password len>:<password>:<roles len>:<roles>:<client's ip addr>:<creation timestamp>:<last access timestamp>:<auth level>

- <auth-db realm name> will be the realm name as configured in the associated authentication database.
- <userid len>:<userid> is the length and the user name of the authenticated user
- <password len>:<password> is the length and the password of the authenticated user
- <number of roles>:<roles> number of roles and the roles assigned to a user
- <client's ip addr>: ip address of the host from where the login was performed
- <creation time>: time when this session was created
- <auth level>: protection level of this session

The <auth-db realm name> field will be used to ensure that the authentication databases of the two web applications are identical. This requires that the various authentication databases associated to web-applications in a SSO environment have the same realm name configured. This approach can result in a conflict when two different WAF's have the same shared secret key and have unrelated web-applications that are bound to two dissimilar authentication databases that have the same realm name configured. It can be seen that this is a highly unlikely scenario that can be safely ignored.

The <userid>, <roles> fields are the fields used for applying access control policies.

The <password len> and <password> will contain the valid values only when the global add-password-in-session-cookie parameter is set to "yes". If the parameter is set to no the values added will be 0 and "NcNoUserPasswd" respectively. The need for this explained later in section 4.3.

AAA will rely on the cookie security configuration for handling cookie replay attacks.

<creation timestamp> is used to ensure that stale SSO User Session Cookies are discarded.

<last access timestamp> is used for staleness due to idle timeouts.

<auth level> will provide administrators the means to hierarchically setup their organization based on protection levels. For e.g the url to view the employee payslip can be configured to be at a higher level than the other urls in a company's finance web site. This field will ensure that cookies generated when trying to access an url at a lower-protection level are not used for accessing an url at a higher protection level. The user must login again to access the higher resource. This is similar to

protection levels of CA Siteminder.

When a NC SSO enabled web-application receives this cookie from the user, the value is decrypted and further processing happens as follows (implementation details of these steps are explained in section 3.3)

1. Check if the authentication databases are identical
2. Check for cookie staleness (creation and idle timeouts)
3. Authentication level checks
4. Apply access control policies using the user name and roles

Single domain SSO configuration will be authentication status is set to “on” and the authentication database bound is of ldap or radius type. Multi domain SSO will require the multi-domain-master and multi-domain-master-url be set.

3.3 Cookie Encryption

The SSO User Session Cookie will be encrypted using the cookie module. The cookie module uses the the cookie encryption key (shared secret) configured for this purpose. It is a necessary condition that all WAFs that contain web applications which will use a common SSO User Session Cookie have the same cookie encryption key configured. Failure to do so will result in the cookie not being accepted throughout.

AAA will also rely on the cookie module for handling cookie replay attacks. This can be controlled using the cookie security configuration.

The cookie module will also decide the behaviour when the cookie encryption key is modified. The assumption here is that the cookie module will continue to decrypt cookies encrypted with the previous key for a certain grace period after the key has been changed. All SSO User Session Cookies generated after the configuration change will be encrypted using the new cookie encryption key.

3.4 NC SSO Checks

This section explains the implementation details of the steps involved in processing the SSO User Session Cookie.

3.4.1 Authentication database validation

The realm name present in the SSO User Session Cookie must match with the realm name of the authentication database associated with the web-application for the cookie to be accepted. If the two fields do not match the cookie will not be accepted and the user will be challenged.

3.4.2 Creation and Idle timeouts

The SSO User Session Cookie contains a creation and a last access timestamp. These timestamps will be used to enforce timeouts based on the configuration fields mentioned in section 3.1.

In order to track the last access timestamp the SSO User Session Cookie must be updated after every request. This default behaviour will be to update the cookie after every 30 seconds. The 'auth-user-session-update-period' configuration parameter can be used to change this behaviour. Typically this would be used in cases where it is considered unnecessary to update the cookie after every request. It is advisable to ensure that the value of this parameter is lesser than the idle and creation timeout parameters.

Please note that the update of the SSO User Session cookie only results in the last access timestamp being updated. The remaining fields remain the same.

3.4.3 Client IP checks

Ensures that the client ip address from where the SSO User Session Cookie is received is the same one from which the login credentials were produced. Cookie module will perform this check for AAA.

The client ip field in the cookie will contain the valid value always. This is irrespective of cookie security configuration, i.e it will be present even if cookie replay checks are turned off.

3.4.4 Authentication level checks

As explained earlier this is used to define a protection hierarchy. Check is performed using the “auth level” field in the cookie and the value of the “authentication-level” parameter.

When the SSO User Session cookie is generated the “auth level” field in it is set to the value of the “authentication-level” parameter of the matched url policy.

3.4.5 Access Control policies checks

Access Control policies on the WAF can be configured using the username (allowed-users), roles (allowed-groups) and allow-any-authenticated-user. The userid and roles field in the SSO User Session Cookie will be used to perform this check.

The roles will be stored as comma separated list. The table below lists some examples

User Roles	Number of roles and Roles field in cookie
eng and qa	2,“eng,qa,”
eng	1,“eng,”
User does not have any roles	0,“”

The trailing “,” is there for convenience, the current AAA C functions work assuming that the trailing “,” exist.

The maximum size of the roles string in the format mentioned above will be 1k. If the roles string exceeds this limit only the first 1k of data will be used and a warning message will be logged.

3.5 Multi Domain SSO

NC SSO will be supported across HTTP domains (e.g www.abc.com and www.xyz.com). NC SSO will ensure that a user after being challenged to provide credentials in one domain will not be challenged again when moving to another domain in a multi domain SSO setup. This is subject to the checks mentioned in section 3.2.2 and explained in section 3.3.

A NC SSO multi domain environment will contain 1 or more master domains and 1 or more slave domains. While it is possible to have multiple master domains, a slave domain can be associated with only one of the master domains. The master domain will play the role of a trusted host that will transfer the SSO User Session Cookie to the slave domains. The following sections will explain the protocol that will be used for transferring the SSO User Session Cookie in a multi domain SSO environment.

The example setup assumes that multi domain SSO is configured for the two HTTP domains www.abc.com and www.xyz.com. www.abc.com is configured to be the slave domain and www.xyz.com is the master. The protocol explanation is split into two parts

1. Unauthenticated user visits the slave domain first
2. Unauthenticated user visits the master domain first

The username in this example is “foobar”

The following table shows the values of multi-domain-master and multi-domain-master-url (refer to section 3.1) for both these web-applications.

Web Application Domain	multi-domain-master	multi-domain-master-url
www.abc.com	no	http://www.xyz.com/ncsso.process
www.xyz.com	yes	/ncsso.process

Please note that the multi-domain-master-url requests are never passed on to the backend server, they are treated similar to the login-processor-path. The multi-domain-master-url in the slave domain must be specified with the protocol field (http, https), the FQDN of the master web-application (www.xyz.com in this case), the port (optional) and the processor uri.

3.5.1 Slave domain first

The user “foobar” tries to access the protected resource <http://www.abc.com/protected.html>. The following sequence of steps happen

- web-application www.abc.com determines that the user is not authenticated (no SSO User Session Cookie)
- web-application www.abc.com also determines that it is in a multi domain SSO setup (multi-domain-master-url is set) and that it is a slave (multi-domain-master is set to “no”)
- redirects the user to the master domain and the redirect url is [“http://www.xyz.com/ncsso.process?NCSSOACT=REQUEST&NCSSOTARGET=http://www.abc.com/protected.html”](http://www.xyz.com/ncsso.process?NCSSOACT=REQUEST&NCSSOTARGET=http://www.abc.com/protected.html)

The NCSSOACT query parameter tells the multi domain processor that the slave domain is requesting for the SSO User Session Cookie of the user. The NCSSOTARGET query parameter contains the url that the multi domain processor will redirect the client to after processing this request. In the current example the user does not have SSO User Session Cookie for the master domain (www.xyz.com).

- multi domain processor redirects the user to <http://www.abc.com/protected.html?NCSSOACT=CHALLENGE>

The NCSSOACT query parameter informs the slave domain that the user is not authenticated and must be challenged to provide login credentials.

- slave domain challenges the user based on the login-method configuration and generates the SSO User Session Cookie for the domain (www.abc.com)
- slave domain redirects the user to the master and the redirect url is <http://www.xyz.com/ncsso.process?NCSSOTOK=<SSOUserSessionCookie>&NCSSOTARGET=http://www.abc.com/protected.html>

The NCSSOTOK query parameter contains the value of the SSO User Session Cookie. This allows the master domain to generate a SSO User Session Cookie for it's domain (www.xyz.com). This step is necessary since there could other slave domains in this SSO setup.

- master domain redirects the user back to <http://www.abc.com/protected.html>

At the end of these steps the user will have two SSO User Session Cookies for the domains (www.abc.com and www.xyz.com).

3.5.2 Master domain first

The user “foobar” tries to access the protected resource <http://www.xyz.com/private.html>. The following sequence of steps happen

- web-application www.xyz.com determines that the user is not authenticated
- web-application www.xyz.com also determines that it is a master in a multi domain SSO setup
- the user is challenged to provide login credentials and a SSO User Session Cookie is set on successful login
- user “foobar” now navigates to the protected url <http://www.abc.com/protected.html>
- the slave domain redirects the user to master, the redirect url is <http://www.xyz.com/ncsso.process?NCSSOACT=REQUEST&NCSSOTARGET=http://www.abc.com/protected.html>

The user “foobar” at this point has a SSO User Session Cookie for the master domain (www.xyz.com) but not for the slave (www.abc.com).

- master domain redirects the user to <http://www.abc.com/protected.html?NCSSOTOK=<SSOUserSessionCookie>>

The SSO User Session cookie value is sent back to the slave using the NCSSOTOK query parameter.

3.5.3 Logout

When a user performs a logout NC SSO will ensure that the user is also logged out of the corresponding master domain.

If the user performs a logout in the master domain no extra action other than removing the master domain's cookie from the user's browser will be taken.

If the user performs logs out from the slave domain, along with removing the slave's cookie the user will be redirected to <http://www.xyz.com/ncsso.process?NCSSOACT=LOGOUT?NCSSOTARGET=http://www.abc.com/nclogin.submit?LOGOUTSUCCESS>

The NCSSOACT query parameter with the value LOGOUT informs the master domain to logout (remove the master 's cookie) the user. The master domain redirects the user to the value of the NCSSOTARGET query parameter.

Please note that logout removes the cookie from the user's browser by expiring it. A malicious user can use a client that ignores the expiry request and keep sending the SSO User Session Cookie for further transactions. WAF will accept the cookie since there is no state maintained on the NC-Gateway to indicate that the user has already logged out. This is not considered a serious threat since timeouts and client ip checks can be used to overcome this problem.

The logout process ensures that the SSO User Session Cookie is removed from the master web application to prevent further propagation of the cookie to other slaves. However the cookie could have been passed to other slave domains before the user performed a logout. This will result in the case where the user still has a valid SSO User Session Cookie for some of the domains in the SSO environment. In this scenario if the user were to leave the web client open, a third person could use the web client to access sensitive information using the previous user's credentials. This problem can be solved either by informing the user to close the browser or to configure a chained logout.

3.5.3.1 Inform the user

As the name suggests the SSO User Session Cookie is a session cookie that will be removed when the user exits from the browser. The default logout success page will be customized to show the following message "It is recommended that you close this browser session for security reasons". Documentation must also be updated to inform administrators to ensure that the logout-success-url configured to a physical url on the backend also contains a message to this effect. This is a very simple and popular method which is known to be effective. This will be the default behaviour of the WAF.

3.5.3.2 Chained Logout

Chained logout is a way by which the user is logged out of all the domains in the SSO environment using HTTP redirects. This requires the administrator to know all the domains in the SSO environment beforehand. Consider the case where three domains www.abc.com, www.xyz.com and www.def.com are a part of a multi domain SSO environment. When a user performs a logout on www.abc.com the SSO User Session cookie of www.abc.com and www.xyz.com are removed by the protocol mentioned above. After these steps have been completed successfully, www.abc.com will redirect the user to logout from www.def.com also.

This can be achieved by configuring the logout-success-url parameter under the authentication container. In this case the logout-success-url in www.abc.com web-application will be http://www.def.com/nclogin.submit?f_method=LOGOUT. This assumes that nclogin.submit is configured as the login-processor-path in www.def.com web-application, if not the url to logout a user will look like http://www.def.com/<login-processor-path>?f_method=LOGOUT. In brief the steps that happen are

1. User performs a logout on www.abc.com and www.abc.com expires it's cookie and redirects the user to www.xyz.com.
2. www.xyz.com expires it's cookie and redirects the user back to www.abc.com
3. www.abc.com redirects the user to perform a logout on www.def.com
4. www.def.com expires it's cookie and redirects the user to www.xyz.com

5. www.xyz.com will simply redirect the user back to www.def.com since www.xyz.com's cookie has been expired in step2. Please note that this step will not result in any error.
6. The SSO User Session cookie of the all three domains have been removed from the user's browser.

This can be extended for more slave domains by simply chaining the logout-success-url configuration

Chained logout will not be automatically done by the WAF since a web-application hosting a domain in a multi domain SSO environment is not aware of all the other domains in the SSO environment.

3.5.4 NCSSOTOK acceptance

The SSO User Session Cookie being accepted by the different domains in a multi domain SSO environment is subject to the checks listed in section 3.2.2 and explained in section 3.3. For e.g if the slave domain does not accept the cookie from the master the user will be challenged to provide login credentials and if the master does not accept the cookie from the slave the request will be silently dropped.

3.5.5 Idle timeouts

In a multi domain SSO environment each domain is responsible for maintaining and enforcing it's own idle timeout. The creation timestamp will be copied from the original cookie transferred using the NCSSOTOK query parameter. This means that the cookie value for the different domains might be different.

4. Open Issues

4.1 Cookie Timeouts

Consider the case where the user has logged into the multi domain SSO environment (user has a SSO User Session Cookie for www.abc.com and www.xyz.com). The user now proceeds to browse through www.abc.com domain for a long period without visiting www.xyz.com. It is possible that when the user moves to www.xyz.com the associated SSO User Session Cookie will not be accepted since it has timed out (idle or creation). This will result in the user being challenged to login again and this is not acceptable when SSO is turned on.

The solution to this problem is to update the master domain's cookie when a slave domain updates it's cookie with new timestamps as explained in section 3.4.2 This update will follow the same protocol as explained in section 3.5.

4.2 POST Redirects

Consider the following case

1. User logs in to a web-application on a WAF
2. Fetches a page that contains a huge form and starts filling it.
3. The time taken by the user to fill the form is longer than the idle or creation timeout and as a result when the user submits the form, he/she is challenged again. This challenge is sent using HTTP redirects.
4. User provides the login credentials again and expects the original POST data to be submitted to the backend application for processing.

The existing WAF implementation does not perform step 4 since the POST data is not saved anywhere when the user is redirected in step 3. Please note that only the url (in this case the form action url) is saved during the redirect.

This issue that will not be addressed in the current release and is marked for future enhancements.

5. Impact on current AAA behaviour

5.1 SSO by default

The WAF will work in SSO mode by default, i.e. the SSO User Session Cookie can be used across web-applications and across WAFs when so configured. This behaviour will be extended to the existing siteminder integration also. Currently

the following options were provided with siteminder

1. Authentication using policy server, authorization on the box (used the stateful AAA method)
2. Authentication and authorization on the policy server (used the stateful AAA method)
3. Authentication and authorization on the policy server and SSO enabled (used the SMSESSION cookie)

Support for 1 and 2 will be dropped, when the authentication database bound is of siteminder type the WAF will use the SMSESSION cookie always. This will ensure that the web-application is working in SSO mode and this is consistent with the new approach.

The status parameter under the siteminder-ssso container will be removed. However, this will not result in any change in the RPC calls or the configuration structures. The configuration daemon (sapd) will always set the mSsoAuth field in AuthPolicy_t structure to SMSSOAUTH when the bound authentication database is of siteminder type.

The same behaviour will be applicable to the planned RSA integration.

5.2 Support for multiple sessions

The default behaviour of stateful AAA was to not allow multiple sessions for the same user within the same realm. A configuration parameter was provided to override this behaviour if required.

In the new stateless model no such restriction will apply, AAA will always allow multiple sessions for the same user within the same realm. This behaviour cannot be changed through configuration.

5.3 Stateful model dropped

Support for the existing stateful AAA model will be discontinued. There will be no session maintained on the box for an authenticated user. The existence of a valid cookie will mean that the user is authenticated and the fields in the cookie will be used for applying access control policies. To the administrators and the users this behavioural change will be transparent. However, the statelessness of AAA will result in some behavioural changes listed below.

5.3.1 Basic Authorization header insert

The basic authorization header contains the base64 encoded form of the following string "<username>:<password>". After the NC SSO changes the password field will not contain the actual password of the user. This means that the Authorization header can only be used for the user id and the password will be a generic string ("NcNoUserPassword").

This behaviour can be changed by setting the global add-password-in-session-cookie parameter to "yes". This will result in the actual password of the user being stored in the SSO User Session Cookie. This setting will be useful in the case where the backend web server is reliant on the WAF sending a Authorization header with a valid password in it. This configuration setting will result in the user's password being compromised if the SSO User Session Cookie is hacked.

5.3.2 Rewrite macros

The password macro (AUTH_PASSWD) will be substituted with the string "NcNoUserPassword" in the default case. This behaviour can be changed by setting the add-password-in-session-cookie parameter to "yes".

The following two authentication rewrite macros will be added

1. **AUTH_COOKIEIP:** Will be replaced with the client ip address as present in the SSO User Session Cookie
2. **AUTH_CLIENTIP:** Will be replaced with the client ip address from where the SSO User Session Cookie was received.

The two macros will have different values when cookie replay checks are turned off and the SSO User Session cookie is sent from a host different from the host where the login was performed.

5.4 State Failover

As stated earlier the current AAA model does not support migration of authentication sessions between cluster machines. This is one of the limitations of the current model.

The new stateless model proposed in this document is designed to address this issue also and it also means there is nothing to be migrated between cluster machines during failover. The SSO User Session Cookie will continue to be valid (subject to checks mentioned section 3) after failover in a cluster.

6. Requirements and Assumptions

6.1 GMT Timestamps

The timestamp entries (idle and creation) in the SSO User Session Cookie will be GMT timestamps. This will ensure that the idle and creation timeouts can be applied correctly across time zones.

6.2 Time synchronization

Time on the all WAFs participating in a SSO environment must be synchronized. Failure to do so will result in timeouts not being applied correctly across these WAFs.

7. Impact on modules

7.1 UI Validations

1. multi-domain-master-url must be set if multi-domain-master is set to 'yes'. This is a conditional mandatory parameter.
2. auth-user-session-creation-timeout and auth-user-session-update period can have a minimum value of 0 and a maximum value of 4294967295.

7.2 Statistics

The following two statistics fields will be added under access-control

1. **invalid-db-cookies:** Will be incremented every time we encounter a SSO user session cookie whose realm entry does not match the realm name of the associated authentication database.
2. **auth-level-low:** Will be incremented every time we reject a SSO user session cookie based on the authentication level configuration. This will also apply for the CA siteminder case.

7.3 New UI commands

None

7.4 Logs

7.4.1 System logs

The condition where the roles string for a given user exceeds the 1k limit a system log will be added with WARNING severity.

7.4.2 Web-Firewall logs

The following new web firewall logs will be added along with this feature

Invalid SSO user session cookie due to auth db realm name mismatch: Attack id **ACCESS_CONTROL_COOKIE_DBMISMATCH** at WARNING severity. This is a new attack id that will be added. There is no solution for this attack id using the policy wizard.

Invalid SSO user session cookie due to lower authentication level: Attack id **ACCESS_CONTROL_COOKIE_AUTHLEVELLOW** at WARNING severity. This is a new attack id that will be added, this will also be reused for the CA siteminder case. There is no solution for this attack id using the policy wizard.

Stale SSO user session cookie due to idle or creation timeout: Attack id **ACCESS_CONTROL_COOKIE_EXPIRED** at level WARNING.

The following two cases will be treated by AAA as no cookie cases, i.e a web firewall log with attack id **ACCESS_CONTROL_NO_COOKIE** at level WARNING will be logged.

1. Tampered SSO user session cookie that could not be decrypted or decoded.
2. Shared secret mismatch

There is an open bug [19101](#) pertaining to this case which when fixed will result in these cases being correctly logged with attack id **ACCESS_CONTROL_INVALID_COOKIE**.

7.5 SNMP

None.

7.6 Upgrade

Upgrade must handle the global parameters reorganization and create the nc-sso container under authentication. It must also remove the status parameter from the siteminder-sso container.

7.7 Development Tools

None.

7.8 Documentation

TBD

8. Future Enhancements

Support for POST Redirects (explained in section 4.2).

9. Implementation Notes

9.1 RPC APIs

The following new RPC calls need to be added

`error_t SetAuthUserSessionCreationTimeout (unsigned int mCreationTimeout);`

The previous RPC will be used for setting the creation timeout, i.e committing the auth-user-session-creation-timeout global parameter. The parameter will be in minutes.

`error_t SetAuthUserSessionUpdateTimeout (unsigned int mUpdateTimeout);`

The previous RPC will be used for setting the session cookie update timeout, i.e committing the auth-user-session-update-period global parameter. The parameter will be in seconds.

`error_t AuthPasswordInCookieSwitch (boolean_t mPasswdInCookie);`

The previous RPC will be used for committing the add-password-in-session-cookie global parameter. The valid value for the parameter is either TRUE or FALSE.

The SetAuthMultipleUserSessionsSwitch RPC is obsoleted.

9.2 RPC Data Structures

AAA configuration data structures will be modified to accommodate new parameters.

The AuthPolicy_t structure will have a added field

```
typedef struct AuthPolicy_t {
    char mRealm[64];
    char mLoginFailureURL[256];
    char mLoginSuccessURL[256];
    char mLogoutSuccessURL[256];
    char mCookieDomain[256];
    char mCookiePath[256];
    char mAuthProcessor[64];
    AuthType_t mAuthType;
    AuthDBType_t mAuthDBType;
    AuthSsoType_t mSsoAuth;
    SmSsoConfig_t mSmSsoConfig;
    NcSsoConfig_t mNcSsoConfig; --> New field
} AuthPolicy_t;
```

Sapd team can reduce the two SSO configuration fields (mSmSsoConfig and mNcSsoConfig) into a single union if they so desire.

The NcSsoConfig_t structure definition will be

```
typedef struct NcSsoConfig_t {
    boolean_t mIsMaster;
    char mMasterUrl[256];
} NcSsoConfig_t;
```

The AuthSsoType_t enum will be modified as

```
typedef enum AuthSsoType_t {
    NCSSOAUTH,
    SMSSOAUTH,
    SSOAUTHTYPEMAX
} AuthSsoType_t;
```

The access control structure will have the extra authentication level field

```
typedef struct AccessPolicy_t {
    AAAAclList_t mAllowedRoles;
    AAAAclList_t mAllowedUsers;
    Boolean_t mAuthSufficient;
    LoginMethod_t mLoginMethod;
    int mAuthenticatationLevel; --> New Field
    char mLoginURL[256];
    char mAccessDeniedURL[256];
    Boolean_t mSendAuthHeader;
    AccessPolicyStore_t mPolicyStore;
    AuthzCacheType_t mCacheAuthzResult;
    int mResourceDepth;
    Boolean_t mIgnoreQueryString;
} AccessPolicy_t;
```

9.3 SAPD commands

None

9.4 File Formats

None.

10. Test Plan

10.1 Dev Test

(List of test cases that will be used to test before delivering to QA. Mention any special requirements such as software

licenses required, any new open-source tools required to enable testing.)

10.2QA Test recommendations

(A list of test cases, in brief, that are recommended for inclusion in the QA test plan. Any ideas, suggestions for test tools. This is *not* a complete test plan, instead, it is a list of things you like to make sure they get into the QA test plan)

11. Future Enhancements

None

12. Alternatives considered

None.

13. Task List

TBD