Cookie-revolver Framework Reference

Version 0.2.5

By
Johan Macedo
johan at gmail dot com

Introduction

About this document

Target audience

This is a technical reference guide for the cookie revolver framework. The reader is expected to have a reasonable understanding of Java, web application security and java web application development.

Structure of the document

The document begins with this the Introduction of the framework, which puts the framework in context and summarizes its functionality.
The next section is the Configuration section, which details how to configure the framework, or install it if you will. This is the section most of interest to the development group which needs to integrate and configure the framework.
The next section is the Administration section, which goes through all the administrative tasks involved to run the framework. As security administration can be laborious, it is good to have an understanding of what is involved to run this framework.
The next section is called Protections and discusses in detail the ways that this framework increases security protections for target systems.
After that section, the BDL reference is provided as a technical guide to writing ban definitions.
Finally, the Dependencies section lists the project dependencies of other open source projects. This is helpful when setting up the framework.

What does the cookie-revolver framework do for me?

- It protects web application by adding another factor to the authentication process
- It protects web application by tracking multiple failed login attempts and preventing brute force attacks
- It protects web application by limiting which user or role can access the system from where

What is the cookie-revolver framework?

It is a java framework that enhances web application security. It provides two-factor authentication for applications that already has one factor (usually memorized password) and integrates with other authorization frameworks, including Acegi.

The high-level features of the framework are:

- Enhances security through two-factor authentication

- Self-selected security questions are used to retrieve cookie certificates
- Supports three kinds of authentications 1) built-in, 2) Acegi-provided and 3) system specific.
- Protects applications through thresholds (for login-failures / security question failures) and lockouts (ip based, user-id based, browser based, system-wide)
- Protections are defined through built in definition language (Ban-Definition-Language) which makes thresholds easily configurable
- In addition to security question usage for certification, also supports security matrix cards.

"How do I test it?": a 15 minute test

This test sets up a functional implementation of the Cookie-Revolver framework for you web-system. It does not integrate with any user/pwd authentication, and is thus not intended for production purposes.

Pre-requisites:

- An existing WAR web-project
- Tomcat or other application server

1. Copy cookierev-x.x.x.jar into a web-inf/lib/ folder
2. Copy the dependent jars in the web-inf/lib/ folder. See Dependencies chapter for more information.
3. Add a servlet filter to your web.xml for the "net.sf.cookierevolver.CRFilter" class and set "/*" as the url-pattern.
4. Add the following init-parameters to the servlet filter:

```
<init-param>
            <param-name>cr.auth.method</param-name>
            <param-value>MOCK</param-value>
</init-param>
<init-param>
            <param-name>cr.policy.default</param-name>
            <param-value>ADMIN</param-value>
</init-param>
```

5. Startup your project
6. Direct your browser to your application. Instead of your first page of the application you will be asked to login. Enter the same username and password (doesn't matter which username and password it is though) and click Login.
7. Enter a security question that the user you just logged in as should answer next time he/she accesses the system from another computer.
8. Enter the answer for the security question and a name for the location.
9. You will now be redirected to the first page of you system.

“How does it work?”: a simple listing of actions and interactions

For most interaction with a CR secured site this is the process

1. Browser sends machine-id cookie
2. User authenticates with username/password
3. Browser sends and gets a new certificate cookie
4. User is redirected to the site main page

The first time a system is used by a browser this is the process

1. Browser gets a new machine-id cookie
2. User authenticates with username/password
3. User answers preset security question or enters matrix card letters
4. Browser gets a new certificate cookie
5. User is redirected to the site main page

“Cookies? How safe is this, and what happens if I clear them?”

Cookies have received a bad rap because they are used incorrectly. This framework assumes that the cookies will be compromised (i.e. people will steal them using Trojan horses and other hacks) from time to time. The cookies are encrypted and they are only one of the elements needed to authenticate.

To be specific, let’s express the functionality in negative terms:

- The cookies used will not let users sign in to the system without a username and password; they will always be required to provide that too.
- The cookies used can’t be copied to other computers, they machine specific and can be made to be IP specific
- The cookies used are not in clear text but encrypted and the password is only available on the server. Further, the cookie does not contain the password of the user so even if a hacker is able to steal the server password, the user password is not compromised.
- The cookies are not a ticket into the system for any user, they are user specific, so even if a hacker gets access to a cookie, they also need to specific username and password that the user of that cookies has.
- A stolen cookie is not valid forever; it will only be usable until the user signs in next time, as a new cookie will be generated at that point.

Further, people will clear their cookies from time to time. The usability of this framework will definitely go down the more they do that. There are plugins to browsers out there that allows you to clear some but not all cookies from your browser, which we recommend people to use. If they don’t, they will still be able to use system protected by the framework but it will be slightly less convenient.

Configuration

Configuration areas

The main configuration of the cookie-revolver framework has to do with which authentication strategy to use.

The framework supports three categories:

- Built-in authentication methods
- Acegi authentication methods
- Custom authentication methods

Depending on the category you choose the setup is different. The basic setting of which to use is the cr.auth.method setting.

The areas that specifically differ for each of these categories are:

- Which settings should be used
- How to integrate the authentication mechanisms with your system
  - What framework pages to link to
  - How to setup GUI
  - How to setup protection/un-protection of you system's pages
- How extendable it is
- Is it production ready
- Is it an enterprise solution

Built-in authentication

There are two built-in methods for authentication: MOCK and DATABASE.

MOCK

The MOCK method is available to allow evaluators of the framework to try it out quickly. It is not meant for production usage. To use the MOCK method, set **cr.auth.method=MOCK**. That is the only required setting.

DATABASE

The DATABASE method stores accounts in the database structure under CR_* tables. It is an authentication method intended for production usage. However, as it is limited in integration with other authentication frameworks, it is meant for stand-alone systems in small and medium businesses, and is not really considered an enterprise solution.
The DATABASE method has uses SHA-256 to hash passwords on the client side using Javascript. This is a highly encourage strategy that makes man-in-the-middle attacks less successful.

ACEGI authentication

Integration

The cookie-revolver framework has classes and support for integration with the ACEGI security framework for Spring.
The support includes the CRAcegiFilter class which is registered with ACEGI in the ACEGI filter chain.
Almost all ACEGI configuration and cookie-revolver configuration is setup in spring configuration files. The cookie-revolver example WAR project includes a complete setup, including ACEGI libraries, spring libraries and spring configuration files.
The cookie-revolver framework JAR file includes a base-spring configuration to be extended by a local acegi.xml file in the target systems WEB-INF folder.
These are the steps to use the cookie-revolver with ACEGI:

- Use **cr.auth.method=CUSTOM**.
- Setup **cr.auth.loginURL** pointing to your ACEGI login page, such as **/acegi-login.jsp**
- Setup **cr.auth.url.prefix** with the prefix of all acegi authentication URLs. You will need to alter some of the setup of basic ACEGI to make this work. The setting is a prefix to make the identification of ACEGI framework URL's faster. An example value would be "/acegi".
- Setup the ACEGI framework, see "http://www.acegisecurity.org/" or the cookie-revolver example WAR project for details of what is required in the spring configuration of the framework.

Technical details of integration

The CRAcegiFilter acts as a filter just like the filters used in the ACEGI framework. It applies the fundamental cookie-revolver checks, i.e. ban-check, machine-identified check, authenticated check, certified check and then passes the call on to the ACEGI framework filters which will apply authentication and authorization.
The AcegiEventListener class is registered in the acegi-common.xml base spring configuration and provides the cookie-revolver framework with details of login attempt information so that the ban-manager can be notified of failed login attempts and the framework be triggered to certify the user after an authentication.
Please note that the cookie-revolver only integrates with ACEGI 0.8.

Custom authentication

Custom authentication is supported to allow system developers to build their own authentication mechanism or continue to use an existing authentication mechanisms while adding cookie-revolver framework to a system to make it strong authentication.
To integrate a custom authentication with the framework both configuration and programmatic integration is required, although both are minimal tasks.

Configuration

The settings required are:

- cr.auth.method=CUSTOM
- cr.auth.logonURL=login.do [i.e. your URL for login functionality]
- cr.auth.logoutURL=logout.do [i.e. your URL for logout functionality]

Programmatic integration

The basic setup is to utilize the interfaces in the root package of the framework, i.e. CRFactory which will return instances of the CRManager and CRConfig. Through the CRManager interface the system code will be able to flag when an authentication failed or succeeded and to return control to the framework for certification. All these methods are accessible on the object returned by calling **CRFactory.getManager().**

recordLoginSuccess(…) and redirectAfterLogin(…)
Calling these methods will change the framework state to allow or disallow the user to get to protected URLs. The second method will redirect to the certification pages of the framework. This call is strictly not necessary as the framework will intercept the next call to display the page anyway. But for convenience it is exposed.

recordLoginFailure(…)
This method will tally up how many times the user has failed at authentication and eventually (based on what ban rules are in effect) lockout the user for a specified number of minutes.

getUserID(…)
The user also has the ability to get out the user name using this method.

isSecure(…)
This method will return true only if the user has authenticated and the login has been certified by a cookie certificate or by the user using security question or matrix card values.

Setting details

The cookie-revolver framework is configured using simple string-based properties. These properties can be placed either "/web-inf/web.xml" as init-parameter values for the filter, as spring <props><prop> for field "config" for the CRAcegiFilter filter bean or in a properties file called "cookierevolver.properties" that must sit in the root package somewhere on the class-path.

Property	Type	Example values	Description
Cr.policy.default	Mandatory	ADMIN RESTRICTED REMOTE_ACCESS	Must be one of the examples listed to the left. This policy will be applied to users that do not match a policy stored in the database.
Cr.bans	Optional	ON 3 failures	BDL expressions of scenarios

		FROM user BLOCK login BY user FOR 15 minutes	when the user should be locked out. Each expression ends with a semi-colon. See BDL reference for more info. By default a 15 minute lockout on 3 login failures is effective.
Cr.auth.method	Optional	MOCK CUSTOM DATABASE	Defines which authentication (user-id/pwd) method will be used. MOCK will make the framework work in demo mode. CUSTOM requires the loginURL property to be set. CUSTOM will allow the system developer to integrate most authentication methods with the framework. DATABASE is the first implementation of IdentityStore authentication strategies. It stores identities in the database, with SHA256 hashed passwords. *The default value is CUSTOM.*
Cr.auth.loginURL	Optional	Login.jsp login.do	Required if authStrategy=CUSTOM. Sets the URL that will be called when a user is not authenticated to process the login.
Cr.auth.logoutURL	Optional	logout.jsp	Required if authStrategy=CUSTOM. Sets the URL that will be called when a user is not authenticated to process the login.
Cr.folder	Optional	cr cookierevolver gk security	The folder that the framework will use to express functionality. This folder should not be used by the application but must be *Default value is "gatekeeper".*
Cr.key.public	Optional	public.key	Names a public key file on the class-path that will be used to encrypt data in the framework.

			The default value is "public.key".
Cr.key.private	Optional	private.key	Names a private key file on the class-path that will be used to encrypt data in the framework. *The default value is "private.key".*
Cr.rolesProvider	Optional	DEFAULT, org.my.RPImpl	Names the implementation of roles provider. The named class must implement the RolesProvider interface. The named class will be used to retrieve a list of roles that are active for a user-id. Using DEFAULT will use the built-in table based provider. *The default value is "DEFAULT".*
Cr.filter.ignore	Optional	/img/* /js/* /introduction.html /*.html	A comma/space/line-break separated list of url-patterns that will not be protected from unauthenticated access.
Cr.config.provider	Optional	mypkg. ExtendedProperties	Points to a class-name which extends Properties class. The properties of this class will be loaded at runtime and overload configuration properties.
Cr.certify.by	Optional	MATRIX QUESTION	Specifies if the framework should use self-selected security questions to certify the user or security matrix cards. *Default value is "question".*
Cr.disabled	Optional	true false	Disables the framework is true. This config property can be set after the filter is initiated but must be set before the filter is called the first time. *The default value is "false".*

Roles provider setup

For systems which has roles mappings to their users, the systems allows an integration interface to read out the roles for the user to make sure the right policy is applied to a user on login. The interface that needs to be implemented is the "net.sf.cookierevolver.ext.RolesProvider" and then register the implementation on the "cr.roles.provider" setting.

This is a very simple interface which returns a String array of role names for a given user-name. The system role-name specific user-names and role-names can then be used when configuring the policies to appropriately map policies to users and roles.
See Policy admin and policy property listings for more details.

Certification process configuration

The cookie revolver framework supports two methods of certification of a new certificate: 1) security questions or 2) matrix cards.

Security questions

Self-selected security questions are one of the ways that the system would verify the user when creating a new certificate for a machine. The security questions are defined on the first login by a user or through the administrate security question admin pages.
This option is encouraged as a simple way of adding new security as people are used to giving having security question in different systems. However, as the security question is just another password, it in some ways reduces "two-factorness" of the solution as it is the same factor as the password, something that the user knows. If this is of concern, the matrix cards are preferable.
To use security questions set **cr.certify.by=QUESTION**.

Matrix cards

Matrix cards are printed of centrally by administrative users. These cards are then given to system users before their first login. They will not be able to access the system without them. The number of letters on the card is configurable.
To use matrix cards set **cr.certify.by=MATRIX**.

Ban threshold definitions

Ban thresholds are defined through the BDL language. The BDL expressions are added to the **cr.bans** setting. The system can handle multiple BDL expressions, each semi-colon separated. A BDL expression defines a threshold and an impact when the threshold is met. Some examples are:

- ON 3 login-failures BY user WITHIN 2 hours BLOCK login BY user FOR 15 minutes
- ON 10 certify-failures BY ip WITHIN 10 hours BLOCK login BY ip FOR 1 hour BLOCK certify BY ip FOR 1 hour

For details of the BDL language please see the BDL reference section.

Database configuration

The cookie-revolver framework database interaction is all taking place through the hibernate framework.
Hibernate requires some configuration, specifically, you need to create hibernate.properties file with connection information OR you can add the hibernate

properties to the cookie-revolver settings and the cookie-revolver framework will forward them when starting up hibernate. See the example WAR project for example of this. The cookie-revolver project does not keep DDL scripts on hand for you to create the tables structure. Please use hibernate **hibernate.hbm2ddl.auto=update** property to generate the schema on the first access.

At this point the framework does not support spring based dependency injected DataSource objects, which would be preferable when using spring. This is coming.

Graphical setup

You will probably want to setup the GUI for the framework to integrate better with your systems look and feel. This is done by creating a **cookierevolver-gui.properties** file in the WEB-INF/classes/ folder. Entries within this file specify HTML code that will be drawn or points to JSP files which will draw the content.

To give an example of how to set it up we will describe how to setup header and footer include files. This is done using **page.top** and **page.end**. Most likely you already have a header.jsp and footer.jsp files in your system that would be setup here, e.g. page.top=/includes/header.jsp and page.end=/includes/footer.jsp. Each page of the framework (login pages, certification pages) will now be displayed with your header and footer.

For more details on other what other GUI properties you can alter, please review the defaultGUI.properties in the net.sf.cookierevolver.gui package in the cookie-revolver JAR file.

Linking to the framework pages

Most of the links to the framework pages will be for admin purpose. All pages are accessible under the url-folder specified in the "cr.folder" setting. The default value is "/gatekeeper" which we will assume you are using for now on.

Here is a quick overview of the framework pages:

Framework page	**URL**	**Usage**
Logout	/gatekeeper/logout	Logs user out and invalidates the session.
Set security questions	/gatekeeper/quest/questions	User re-set security questions functionality
Clear all cookie	/gatekeeper/clear	Removes the users cookies and super-certificate.
Administrate policies	/gatekeeper/policyadmin/select	Admin function of policies
Generate super certificates	/gatekeeper/supercert	This function is provided so that super certificates can be generated centrally so that an administrator

		can generate it and send it out to regular user.
Set super certificate.	/gatekeeper/cert/super	This function will allow an administrator to set a super cert for the current browser. For a regular user, a textinput where they can paste a super certificate ID is provided.
Administrate accounts	/gatekeeper/auth/admin	Admin function for user accounts, only used for DATABASE auth method.
Remove bans	/gatekeeper/removebans	Admin page to remove active lockouts.

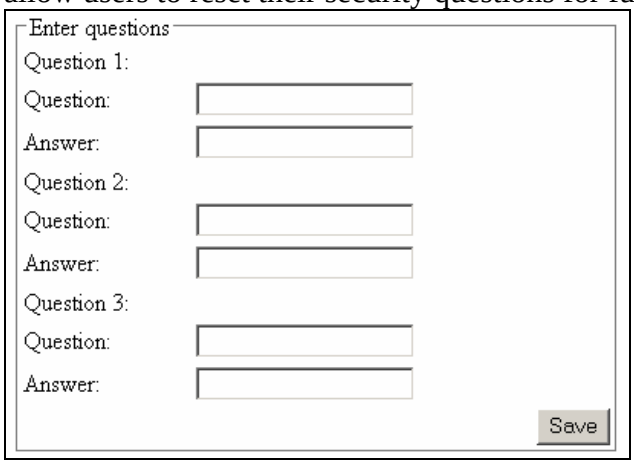
User pages

Logout

The logout URL is **/gatekeeper/logout**. The result when navigating to this page is that the session data will be erased and the user will have to re-authenticate to use the system. This page should be linked to from within your system. The user must be authenticated to access this page.

Set Security Questions

The set security question page URL is **/gatekeeper/quest/questions**. This page will allow users to reset their security questions for future certifications.



This page is only applicable if you are using the "cr.certify.by=question" (which is default). If you are using the matrix card certification, the user cannot generate a new matrix themselves, but have to as an administrator to create it for them (see Administrate matrix page below).

This page should be linked to from within your system. The user must be authenticated to access this page.

Clear cookies
The url for this page is **/gatekeeper/clear**. Accessing this page will clear cookies for the current user and super-certificate cookies. This is usefull is you are testing the framework but does not need to be exposed to the end users.

Admin pages

Administrate policies
This is the most important of admin pages. It is access through **/gatekeeper/policyadmin/select**.
The policy administration pages are used to setup the different policies and add new policies. See policy property listing for details on what each of the policy fields mean.
A policy can be associated with a specific user or a role. If DATABASE auth method is used the built in user administration system allows you to register roles for users. If you are using CUSTOM auth method, then the RolesProvider interface needs to be implemented and registered with cr.roles.provider setting to allow the framework to access your systems user-to-role mappings.
Once the mapping is available, the system will resolve the role list for a user and find all policies that match the user-name and roles for the user and the policy with the highest "priority" property will be loaded as the active policy for the user.

Remove bans
The "Remove bans" framework page allows administrators to remove (i.e. disable, disarm) bans/lockouts of users, IP's or machines.
If a system ban is in effect, this page will obviously not be available and everybody must wait for the ban to disable by itself or the system admin can restart the system.

Generate super certificate
This page allows administrators to generate super certificates. This is available so that they can be generated centrally and sent by e-mail to locations where the administrator will not visit physically.
The super certificate is a 32 character string.

Enter super certificate
This is used either by the end user or an administrator to activate a super certificate for the current machine.
If you are an administrator this screen will you to simply click "Set super certificate" which will generate a new certificate and apply it to the machine.

Administrate matrix
If the systems is setup to use matrix cards for certification, this page allows administrators to generate the matrix cards. Simply enter a comma, tab, enter or space separate user list and a page with new cards with generated cards will be displayed for the administrator to print, cut and laminate for the user.

Administrate security questions

If the system is setup to use security questions for certification, this page allows administrators to reset users security questions. Security questions should really be self-selected and it is not really recommended to use this page but could be used if the user forgets their answers, and an administrator resets them to something temporarily.

Administration

The administration of CR involves maintaining user Security Questions, Policies, Certificates, Bans and other Authentication configuration related maintenance.
Policies define what different users and user-roles can do and for how long.
Certificates maintenance is done only on occasional basis where certificates are adjusted/created on user-request. There is something called super-certificates that are not user-specific and can be installed at trusted locations to allow multiple user-account access without user needing to answer security questions.
Bans prevents user-access and take effect automatically under predefined states but has manual overrides to disable them, which involved maintenance.

Certificates

Certificate administration involves generating super-certs. Super certs are highly trusted certificates that can be installed on trusted locations.
Trusted locations means shared workstations that are located in a trusted (i.e. secure, watched) area. These will be allowed to have multiple people accessing the system without ever having to answer security questions. This is in some ways a mechanism of side-stepping the benefits of the framework, and should be used with caution. The machines that have these certificates needs to be locked down (both system-wise and hardware wise) and considered high-risk as they are in essence an open gate into the system.

Policies

The maintenance of policies is the most involved area of administration for the cookie-revolver framework. With policies the administrator defines who can access the system from where by setting the remote-access flag. Remote access means that the user can create new access points for the system. If a user does not have remote access he/she can only login from a machine that is considered a Trusted location, see more about this below. Policies also defines how many times a user can login before needing to answer a security question again. It also defines the maximum time between logins at a location. It defines if there is IP restriction on the location, i.e. if the IP must match an IP range or is static and must be the same after the first registration of a location.

Table of policy record settings:

Field	**Description**	**Example values**
staticIP	If true, the IP of the location must remain the same post the initial creation.	True False
remoteAccess	If true, the user can login in a new locations and answer security questions to create the first certificate.	True False
Priority	Defines which policy will take effect if a user has multiple policies defined for them.	1 10 1000
usageTimesBeforeReverify	Defines how many times a location can be logged on at before	100

	the user must answer a security question again.	1000
maxTimeBetweenUsage	Defines the maximum amount time that can elapse between usages of a location by a user. **DEFINED IN MINUTES.**	1440 14400
expireCookie	Defines the maximum amount time that users can use a certificate until the user need to answer security questions again. **DEFINED IN MINUTES.**	14400 144000
ipFilter	A REGEXP expression that the users IP must match to be allowed to access the system. Regular path like IP filters are also supported, such as 192.168.*.* or 205.1.0.*.	
certsMax	The maximum number of different locations that can be used by the user to access the system.	3 10
certsCurrent	A counter of the number of locations currently using this policy. When this number reaches the certsMax, the policy will not allow further creations of locations.	0
generateSuperCerts	Defines that this user can generate super-certificates that can be used to create trusted locations.	True False
administratePolicies	Not used currently. This will control access to the policy listing through the system.	True False
userID	If this is provided, the policy is user-specific.	user1 johan
Role	If this is provided, the policy is role specific.	role1 superuser
default_answer	This is an interim development property that should not be used in production systems. It allows users to access the system without being pre-registered with security questions. Normally, if there are no security questions registered for a user, the user cannot access the system.	test

Bans

When bans take effect, the user(s) will not be able to login into the system. This can be highly inconvenient and an administrative interface is available to remove active bans.

Security questions

Security questions are needed for a user to login in from at new locations. It is only required when users have remote access. Without remote access, should never have to answer security questions.
The URL for administration is /gatekeeper/quest/adminQuestions. The answers to the questions are hashed and stored in the database.
The user can self select their security questions but the administration feature allows a central security administrator to override these settings.

Authentication configuration related maintenance

Administration of Authentication pieces is dependent on what configuration setting is used in authStrategy setting. The MOCK strategy requires no administration. The CUSTOM strategy is non-specific to the framework and we can't help you there.

The DATABASE strategy has an administrative interface where users can be added, disabled/enabled, passwords expired/unexpired, accounts locked-out/let-in-again and passwords reset. The URL for the administration is /idstoreauth/admin.

Protections

Basic flow

- Machine identification and IP/machine ban clearance
- User identification and user ban clearance
- User-With-Machine certification

Basic protections

- Two factor authentication:
- Username/Password – something you know
- One-time encrypted cookie – something you have
- No resources in the protected system can be accessed without all steps in the authentication process are passed successfully.
- Initial encrypted cookie is retrieved through answering previously self-selected security question or by entering matrix card letters.
- Username/password authentication is also required prior to retrieving the cookie.
- The next time the user access the system, the encrypted cookie is sent to the server. If it is valid the server sends back a new encrypted cookie.

Overview

This section describes in detail how the different areas of the framework works together to protect your system.

- The physical mechanisms can be divided into:
- Basic framework protections
- Failure/Banning mechanisms
- Policy based protections

Failure / Banning mechanisms

Definition of Ban, Threshold and Failure

Bans are items that prevent access to the system. They can be specific to a property of the access, like Machine-id, IP or User-id. It can also be general, which is called a System ban.
Bans are in effect for a predefined period of time.
Bans are triggered when a number of failures (login failures/security question failures/certificate failures) happen within a specific time-period. The thresholds are different for different entities.
For example, for users, three failed user-logins could lock that account for 30 minutes.
Or, for IP's, 10 failed user-logins will ban all access from that IP for 30 minutes.
The entities that can be controlled are user-account, machine (location), IP, and system.
System means the system as a whole. For instance, if there are more than 100 user-login

failures within 2 hours, the system can go into lock-down mode not allowing any user-logins for 60 minutes.
Bans are triggered through defined thresholds. A threshold is defined as a number of errors/failures during a specific period of time. The errors/failures do not need to be consecutive.
The thresholds can be limited to certain kinds of failure but can also be configured to include all failures. A typical failure is the login-failure. Another example is the certificate-cookie validation failure. A third one would be security question answer failure.
Failures are tracked individually for different criteria, specifically IP, Machine-id and User-id.

IP bans

All incoming calls can be blocked on the basis of IP. An IP ban is most commonly triggered through multiple login-failures from an IP.

Machine bans

All incoming calls can be blocked on the basis of machine-id. A machine ban is most commonly triggered through multiple login-failures from a machine.

User bans

All incoming calls can be blocked on the basis of user-id. An user-id ban is most commonly triggered through multiple login-failures from an user-id.

System bans

To protect the system from being flooded with requests from different IP, machine-id and user-id there is a system-ban level.
This ban has a far higher threshold than the other bans, not to trigger easily. When a system ban is in effect, no system access is possible.

Basic framework protections

Resource protection

All URL's that have not been defined as unprotected. The protection is created through the mapping of the CRFilter which should be mapped to /*, i.e. all URL's. Resources can be unprotected by configuring the framework to ignore security check for access to URL-patterns (such as *.gif, /images/* etc).

Flow control

The framework controls the flow through the authentication process. The framework will kick in on the first call to URL that is protected. The framework will first retrieve or assign a machine-identifier cookie. This is simply a 15-character long random GUID which will is used identify the location. This GUID is not encrypted but is later baked into the encrypted certificate-cookie. A hacker will be redirected back the current step in the process, if URL's that represent later steps are called directly. I.e. steps can not be bypassed.

Initial Certificate retrieval

The first time a user access the system from a specific browser on a PC, the user must answer a previously self selected security question. The question is not asked until the user has passed step 1 and 2.

Certificate data checks

Certificates are specific to both a user and a machine. They will only be accepted if they the stored machine-id and user-id matches the identified machine and the user-name/password used to login.
So if a hacker steals a certificate-cookie from a computer, the hacker must get the machine-id cookie as well, and get user-name and password for the user that who initiated the certificate.
As part of the certificate is a timestamp which will change for each interaction with each exchange of certificate-cookies.
If any of the values in a certificate-cookie submitted are different than the latest certificate-cookie on file, the user must retrieve a new certificate by answering the security question again.

Encryption

The certificate cookie is encrypted. This will make it a lot harder for a hacker to generate fake certificates, as he/she would have to gain access to the server encryption key.

Super certificates

Security Administrators of the system can generate super certificates. Super certificates are not user specific. These certificates can be installed at trusted locations as a way of making them continuously connected to the system. These certificates can be configured to have longer expiry periods.

BDL reference

BDL stands for Ban-Definition-Language and is a language which expresses thresholds and impacts.

Syntax legend:

[xyz_expression] – Represents a named expression
(xyz) – The content within the brackets are optional and has default values.
- Represents a number
ABC / abc – Represents a key word. All keywords are key non-sensitive.

[bdl_expression]

The first part of a BDL expression defines the condition that will trigger the impact. The second part defines the impact. The impact part can be repeated to specify multiple impacts.

Syntax: [condition_expression] [impact_expression]

Example:

- ON 3 login-failures BY user WITHIN 2 hours BLOCK login BY user FOR 15 minutes
- ON 3 login-failures BY user WITHIN 2 hours BLOCK login BY user FOR 15 minutes BLOCK login BY machine FOR 5 minutes

[condition_expression]

A condition expression defines what must occur and during what time the condition will be tracked.

Syntax: [event_expression] ([within_expression])

Example:

- ON 3 login-failures FROM user WITHIN 3 hours
- ON 2 failures FROM ip

[event_expression]

The event expression defines what event is tracked and for entity (system, ip, machine or user):

Syntax: ON # ([action]-)failure(s) (FROM [entity])

Where [action] is either “login” or “certify”.
Where [entity] is one of “system”, “machine”, “user” or “ip”.

Examples:

- ON 2 login-failures FROM ip
- ON 1 failure FROM user
- ON 3 security-question-failures

[within_expression]

Defines the time period that will be watched for occurrences of the failures.

Syntax: WITHIN [period_expression]

Examples:

- WITHIN 1 hour
- WITHIN 2 days, 5 minutes
- WITHIN 3 hours

[period_expression]

Defines a period of time. The expression syntax can be repeated, separated with commas.

Syntax: # [type]

Where [type] is one of “day”, “minute”, “hour”, “second”, “week” or “year”. All the types can optionally have an extra “s” after. In addition, “sec” and “min” is also supported.

Examples:

- 2 days
- 2 hours, 5 seconds
- 1 day, 2 hours, 5 minutes

[impact_expression]

Expresses the impact once the threshold is met. This statement can be repeated multiple times sequentially to identify

Syntax: BLOCK [action] BY [entity] FOR [period_expression]

Where [action] is either “login” or “certify”.
Where [entity] is one of “system”, “machine”, “user” or “ip”.

Examples:

- BLOCK security-question BY user FOR 30 minutes
- BLOCK login BY user FOR 15 minutes BLOCK login BY ip FOR 5 minutes

Dependencies

The cookie-revolver framework has several dependencies. All dependencies are mandatory, except for the ACEGI libraries and log4j.

Filename	**Description**	**Reason**	**License**
jug-lgpl-2.0.0.jar	Java Uuid/guid Generator (JUG)	UID generation	LGPL
hibernate3.jar	Hibernate	Persistence	LGPL
cryptix-jce-provider.jar	Cryptix JCE provider	RSA encryption/decryption	BSD
bouncycastle-shaonly.jar OR other bouncy castle distro	Bouncy Castle SHA classes	SHA hashing	BSD style
commons-logging-1.0.4.jar	Apache Commons logging	Logging	Apache
ehcache-1.1.jar	EH Cache	Hibernate dependency	Apache
Dom4j-1.6.1.jar	Dom4j	Hibernate dependency	BSD style
concurrent-1.3.2.jar	Apache Concurrent	Hibernate dependency	Apache
commons-collections-2.1.1.jar	Apache Commons collections	Hibernate dependency	Apache
Cglib-2.1.3.jar	CGLib	Hibernate dependency	Apache
asm.jar	ASM	Hibernate dependency	BSD
antlr-2.7.6rc1.jar	ANTLR	Hibernate dependency	BSD
Acegi-security-0.8.3.jar	ACEGI for spring	Acegi framework (Optional, needed if using ACEGI)	Apache
oro-2.0.8.jar	Apache ORO	Acegi dependency (Optional, needed if using ACEGI)	Apache
spring-1.2-RC2.jar	Spring framework	Acegi dependency (Optional, needed if using ACEGI)	Apache
log4j-1.2.11.jar	Apache Log4j	Logging (Optional)	Apache