Ring Authentication Middleware

David Jagoe, Rheo Systems

October 12, 2011

Revisions

Revision	Date	Description	Author
1.0	2011-09-27		David Jagoe
1.1	2011-09-28	Various design fixes - see log	David Jagoe

1 Introduction

Every request to the application must specify user identity in some way:

- As part of the request: id=1234 & key=4321, or
- Posting credentials to the login URL
- Requesting to become anonymous by hitting the logout URL
- By session id - identifying an established login session
- None of the above, in which case the user is anonymous

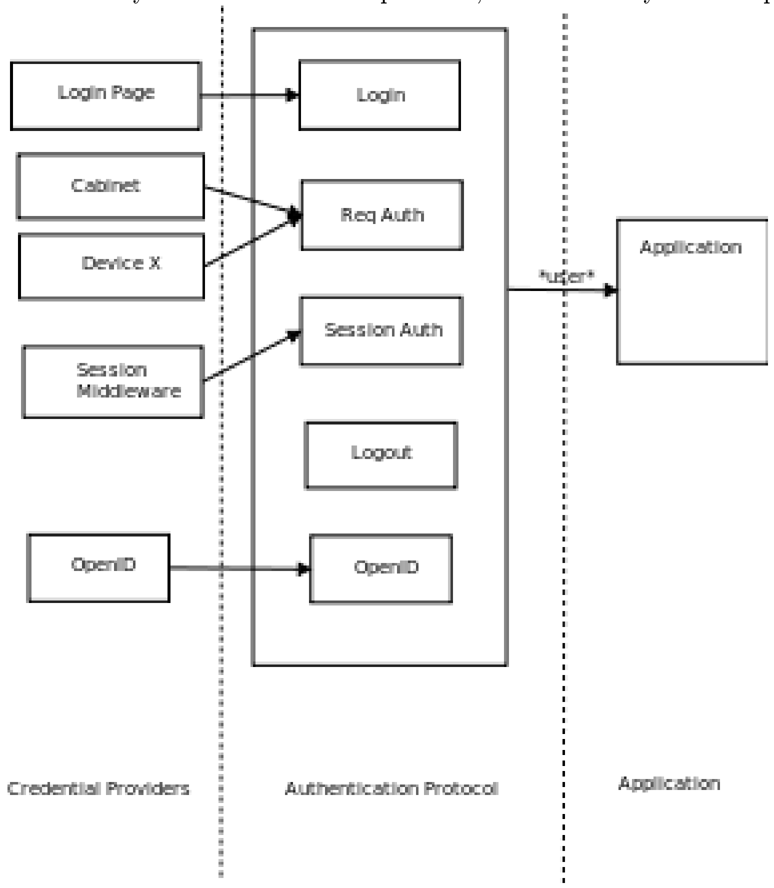
The user identity is made available in a uniform way for use by the rest of the application. The application may refer to user identity for a number of reasons: e.g. to determine authorization or user-specific behaviour (e.g. customizations). The application should be unaffected if a new authentication method (e.g. OpenID) is required.

2 Terminology

Authentication protocol The means of user authentication and the definition of communication between this module and the credential provider. Possible protocols include:

- HTTP login
- AJAX login
- OpenID
- Active login session
- Per-request authentication
- Logout (an "unauthentication" method which can be treated in this document as an authentication method)

Credential providers An application that uses this module must provide means of capturing user credentials from the end user and suppling them to this module via the corresponding **authentication protocol**. In other words each credential provider will use exactly one authentication protocol, but there may be multiple users of the same protocol in a single application.



3 Purpose

The purpose of this module is to hide the details of user authentication from the using application. It is not the purpose of this module to maintain a list of URLs that are protected and therefore require login, or to handle redirection to a “next_url” on login or logout. These items would be the responsibility of the application or some other component.

4 Limitations

Assumptions

- The application is fundamentally an HTTP application, and as such the outer-most interfaces are HTTP requests and responses

Compatibility & Dependencies

- This module is compatible with ring, which has a specific format for HTTP request and response maps
- The module also depends on ring.middleware.session, ring.middleware.params and ring.middleware.keyword-params
- The durability of login session (e.g. between server restarts) depends directly on the durability provided by session management

Extensibility

- Extending the authentication mechanism (e.g. 2-factor authentication, OpenID, Ajax login box) will not affect the application code - which should just rely on *user*. This design is extensible by adding new authentication protocols that do not affect existing protocols. Adding a new protocol would involve updating this document to add the corresponding cases and reviewing for completeness.

5 Specification

5.1 Inputs

Input	Short	Type	Comment
uri	uri	URI	The URI of the current request
method	meth	Enum {GET, POST, ...}	The HTTP request method
no_expire	nexp	Bool	Contained in the session
time_of_last_request	tolr	Long	Contained in req session, ms since epoch
logged_in_user	liu	User	Contained in the req session
username	un	String	Contained in POST parameters
password	pw	String	Contained in POST parameters
remember_me	rem	Bool	Contained in POST parameters
user_id	uid	String	Contained in the GET params
user_key	key	String	Contained in GET params
user_repository	udb	UserRepository	
logger	logger	Logger	

5.2 Outputs

Output	Short	Type	Stateful?	Comment
user		User	no	dynamic variable
time_of_last_request	tolr	Long	yes	element in the session
logged_in_user	liu	User	yes	stored in the session
no_expire	nexp	Bool	yes	stored in session
flash_message	*msg*	String	no	dynamic var
log_msg	log	String	no	Written to logger

5.3 Parameters

Parameter	Type
login_url	URL
logout_url	URL
timeout_ms	Integer > 0
messages	Map

messages default to {:login-success (fn [req] "Welcome") :login-failure (fn [req] "Incorrect credentials") :logout (fn [req] "Bye")}}

5.4 Valid Input/Output Combinations

Inputs

Protocol	**Inputs**									
	'uri	**'meth**	**'rem**	**'tolr**	**'liu**	**'un**	**'pw**	**'uid**	**'key**	**'udb**
login	{login_url}	POST	<BOOL>			<Str>	<Str>			<Rep>
logout	{logout_url}	GET								
session				<Long>	<User>			-	-	<Rep>
request								<Str>	<Str>	<Rep>

Outputs

Protocol	**Outputs**					
	***user*'**	**tolr'**	**liu'**	**nexp**	***msg***	**log**
login	<User>	<Long>	<User>	<Bool>	<String>	<String>
logout	<User>	nil	nil	nil	<String>	<String>
session	<User>	<Long>	<User> or nil	<Bool>	nil	<String>
request	<User>	nil	nil	nil	nil	<String>

In the tables:

- "-" means
  - in the case of input: that the input must not be provided in this case
  - in the case of output: that the output does not make sense for this case and is not changed
- blank means the value is not used, even if it is set; note that the table therefore defines login, logout, session, request based on inputs. They will be more rigourously defined below.
- a type (in angled brackets) means that the value is used/set (and is a variable)
- an entry which is not a type means that specific value must be provided (or is set) in that case
- a value in braces is the value of parameter

5.5 Outputs as Functions of Inputs

liu(T) =

login	f('un, 'pw, 'udb)
logout	nil
session	f('liu, 'tolr, 'rem, 'udb)
request	-
bad_req	nil

***user*(T)** =

login	liu(T)
logout	anon
session	liu(T)
request	f('uid, 'key, 'udb)
bad_req	anon

***msg*(T)** =

login	f(liu')
logout	messages(:logout)
session	-
request	-
bad_req	-

log(T) =

login	f('un, 'pw, 'rem, liu')
logout	f('liu)
session	f('liu, 'tolr, 'rem, 'udb)
request	-
bad_req	f(ALL_INPUTS)

tolr =

login	System/currentTimeMillis()
logout	nil
session	System/currentTimeMillis()
request	-
bad_req	-

nexp =

login	f('un, 'pw, 'rm, 'udb)
logout	nil
session	nil
request	nil
bad_req	nil

Note that while *user* and liu are similar they represent orthogonal concepts

1. liu is an interface to the browser (session), while *user* is an interface to the application
2. We must deal with login session not existing at all - it doesn't make sense to store an anonymous user in the session (which may not exist)

3. liu identifies the presence or abscence of a *login session*; *user* identifies the user active in the current request which is independent of login session

5.6 TFM

5.6.1 liu(T) =

	login ∧	**session ∧**	**(logout ∨ request ∨ bad_req) ∧**
id_match	auth_login	auth_session	nil
¬id_match	nil		

5.6.2 *user*(T) =

	login ∧	**session ∧**	**request ∧**	**(logout ∨ bad_req) ∧**
id_match	liu_or_anon		auth_request	anon
¬id_match	anon			

5.6.3 tolr(T) =

login	curr_time
logout	nil
session	curr_time
request	-

nexp =

login	id_match ∧ pw_match ∧ rem(r(T))
logout	nil
session	nexp(r(p(T)))
request	nil
bad_req	nil

5.6.4 *msg*(T) =

login ∧		**logout ∧**	**(session ∨ request ∨ bad_req) ∧**
success	**¬success**		
messages(:login)	messages(:login_fail)	messages(:logout)	-

5.6.5 log(T) =

login		**logout ∧**	**request**		**bad_req**	**session ∧ ¬auth_session**
success	**¬success**		**success**	**¬success**		
info_in_success	info_in_fail	info_logout	info_request_sucess	info_request_fail	warn_bad	info_unauth_session

5.6.6 Definitions

login(T) = $\quad$ url(r(T)) = login_url ∧ method(r(T)) = POST∧ un(r(T)) ϵ String ∧ pw(r(T)) ϵ String

logout(T) = $\quad$ url(r(T)) = logout_url

session(T) = 'tolr(r(T)) ϵ ℝ ∧ 'liu(r(T)) ϵ User ∧ ¬(login ∨ logout)

request(T) = 'uid(r(T)) ϵ String ∧'key(r(T))) ϵ String ∧ ¬session

bad_req(T) = ¬(login ∨ logout ∨ request ∨ session)

id_match(T) = [1]

login	{udb::find-login-account('un(r(T))}≠ nil
logout	false
session	{udb::find-login-account('liu(r(T))} ≠ nil
request	{udb::find-active-account('uid(r(T))}≠ nil

auth_login(T) =

pw_match	udb::find-login-account('un(r(T)))
¬**pw_match**	nil

auth_request(T) =

pw_match_s	udb::find-active-account('uid(r(T)))
¬**pw_match_s**	anon

auth_session(T) =[2]

¬udb::login?(username('liu(r(T)))			anon
udb::login?(username('liu(r(T))) ∧	'rem(r(T))		udb::find-login-account('un(r(T)))
	¬'rem(r(T))	expired	anon
		¬expired	udb::find-login-account('un(r(T)))

liu_or_anon(T) = liu(T) if liu(T) != nil else anon

_pw_match(id, pw) = udb::verify_password(id, pw)

pw_match_s(T) = _pw_match(uid(r(T)), key(r(T)))

pw_match(T) = _pw_match(un(r(T)), pw(r(T)))

anon(T) = 'udb::get_anonymous()

expired(T) = (curr_time() - 'tolr(r(T))) > timeout_ms[3]

curr_time(T) = {pragma/clojure: (System/currentTimeMillis)}

info_in_success = curr_time(T) + ": User " + un(r(T)) " logged in successfully"

info_in_fail = curr_time(T) + ": User " + un(r(T)) " failed to log in by password"

[1]User is an Identity

[2]can't just find_active because that would result in nil, not anon

[3]tolr always a number when used - see definition of "session" and verify that this function only used under that condition

info_logout = curr_time(T) + ": User " + liu(r(T)) " logged out"

info_request_success = curr_time(T) + ": User " + uid(r(T)) " authenticated successfully"

info_request_fail = curr_time(T) + ": User " + uid(r(T)) " failed to authenticate"

info_unauth_session = curr_time(T) + ": User" + 'liu(r(T)) + " has been logged out"

warn_bad = curr_time(T) + "Bad auth req: " (str ALL_INPUTS)

6 System State Transitions

In order to identify internally maintained states we identify that the following variables hold state:

- liu
- tolr
- udb
- rem

Definition of module states:

1. User logged out (this must be the starting state)
   (a) liu = nil
2. Active login session
   (a) liu != nil

6.1 State Transitions

From\To	logged_out	logged_in
logged_out	$\neg$(login $\wedge$ id_match $\wedge$ auth_login)	(login $\wedge$ id_match $\wedge$ auth_login)
logged_in	logout $\vee$ bad_req $\vee$ request $\vee$ $\neg$(expired $\wedge$ $\neg$rem)	$\neg$(logout $\vee$ bad_req $\vee$ request $\vee$ (expired $\wedge$ $\neg$rem))

7 User Interface / Usage

Note that this is NOT a specification - it merely shows you how to use the feature and supply the parameters. You need to know the structure of Ring req and resp maps in order to properly use this interface.

7.1 REPL

```
(def auth-config
  {:login-url "/login"
   :logout-url "/logout"
   :timeout-ms 1000})

(defn my-handler [req]
  (println *user*))

(def auth-handler (with-authentication my-handler user-repository auth-config))

;; per-request authentication
;; ----------------------------

(auth-handler {:params {:user-id "12345" :user-key "abcde"}})
;; -> {:username "12345" :password "somehash" ...}

(auth-handler {:params {}})
;; -> {:username "Anonymous" :password "somehash" ...}

;; session-based authentication
;; ----------------------------
(auth-handler {:session {:logged-in-user {:username "12345"} :time-of-last-request (now)}})
;; -> {:username "12345" :password "somehash"}

;; login
;; -----
(auth-handler {:uri "/login" :params {:username "david" :password "foo"}})
;; -> {:username "david" :password "somehash"}
```

7.2 Ring Middleware

The authentication module is designed to be used as ring middleware. It can be hooked up in the normal way. The application is expected to implement a login page that captures username and password and causes a post to {login_url}. Normally this page would be implemented as handling a GET on {login_url}. Normally the application would also implement a handler for POST {login_url} which would redirect based on the value of *user*.

8 Implementation

8.1 Performance

- What is the performance impact of hitting the database on every request to check if the user is still active?
  - If its a problem could we use memcache here... it seems like the perfect candidate because users are so isolated from the rest of the system and easy to flush if they are disabled or otherwise changed.

9 Implementation Review

Need to set session to nil on logout & expiry			

10 Design Review

Query	**Commit**	**Response**	**Change**
Should liu and *user* be so similar?	6a63595c	1	Doc change: make distinction clear
Rename stateless to sessionless	6a63595c		Doc change
Do not allow devices to log in		2	Design change
Login/logout multiple times?	6a63595c	3	No change
User provides URL auth when logged in?	6a63595c	4	No change
Provide a state transition diagram	6a63595c		Added state transition table
How difficult to add OpenID support?		5	

1. We need both concepts and they are similar but need to be different because they represent different things, the distinction has now been made clear above.
2. A piece of equipment should not be able to establish a login session. We will follow Amazon's model: a "user" has an identity (username) and password regarless of whether that user will log in (human) or authenticate per request (human/device).
   (a) a user may *either* log in *or* use request auth but not both
   (b) the design change reflects this by pushing responsibility to the user repository: we must be able to identify whether users can have login sessions and whether or not their accounts are locked.
3. At the moment a user who is logged in is allowed to post new credentials to the login url with the result that the new user supplied would become logged in. This is acceptable and it is up to the application to decide how to present (or indeed disallow) this behaviour. For example this can be presented as "switch user" which implies that you don't need to log out first. Log out is even simpler - I have reviewed the design and the user may attempt to log out whether there is an active login session or not.
4. This is considered a "feature" of the design: a login session can be over-ridden for a single request by supplying different credentials. This has no negative effects and while it may not be used in practice it also does not complicate the design.
5. We need to be able to add OpenID / Facebook login support later without affecting the application or rewriting the authentication module. How will support for this be added?
   (a) Added a new point on extensibility and made the concepts of credential suppliers and authentication protocols explicit. Added OpenID support simply means adding a new authentication protocol.