

Securing Clojure web services & applications with Friend

Chas Emerick

@cemerick

<http://cemerick.com>

What are we not talking about?

- SQL injection
- Cross Site Request Forgeries
- Replay attacks
- Untrusted code evaluation/jailing

What are we talking about?

- Authentication (a.k.a. A1, authN)
 - “Who are you?”
- Authorization (a.k.a. A2, authZ)
 - “What are you allowed to do?”

Auth options in Clojure-world

- spring-security
- brentonashworth/sandbar
- remvee/ring-basic-authentication
- DerGuteMoritz/clj-oauth2
- Roll-your-own

None were completely fulfilling...






































Seriously, why?

- Implementing auth (properly) is:
 - High risk
 - Minimal reward
 - Absolutely necessary
 - *Should* be a solved problem

Envy

<https://github.com/bnoguchi/everyauth>

 Password	 Github	 Dwolla	 mixi
 Facebook	 Instagram	 OpenStreetMap	 Mailchimp
 Twitter	 Foursquare	 VKontakte	 Meetup
 Google	 Yahoo!	 Mail.ru	 Mendeley
 Google Hybrid	 Justin.tv	 Skyrock	 Smarterer
 LinkedIn	 Vimeo	 Gowalla	 RunKeeper
 Dropbox	 37signals	 TriplIt	 Box.net
 Tumblr	 Readability	 500px	 OpenId
 Evernote	 AngelList	 SoundCloud	LDAP (experimental)

Wishlist

- Assume Ring
- Use any authentication authority I want
 - Easy: username/password, HTTP Basic
 - PITA: OpenID, OAuth(2)
 - Custom: multi-factor auth, phishing prevention
- Flexible authorization options
 - role-based
 - room for ACLs, capability systems, & more

Friend

<http://github.com/cemerick/friend>

- Ring middleware
- Authentication workflows are Ring handlers++
- Credential sources are functions
- Hashing functions are...functions

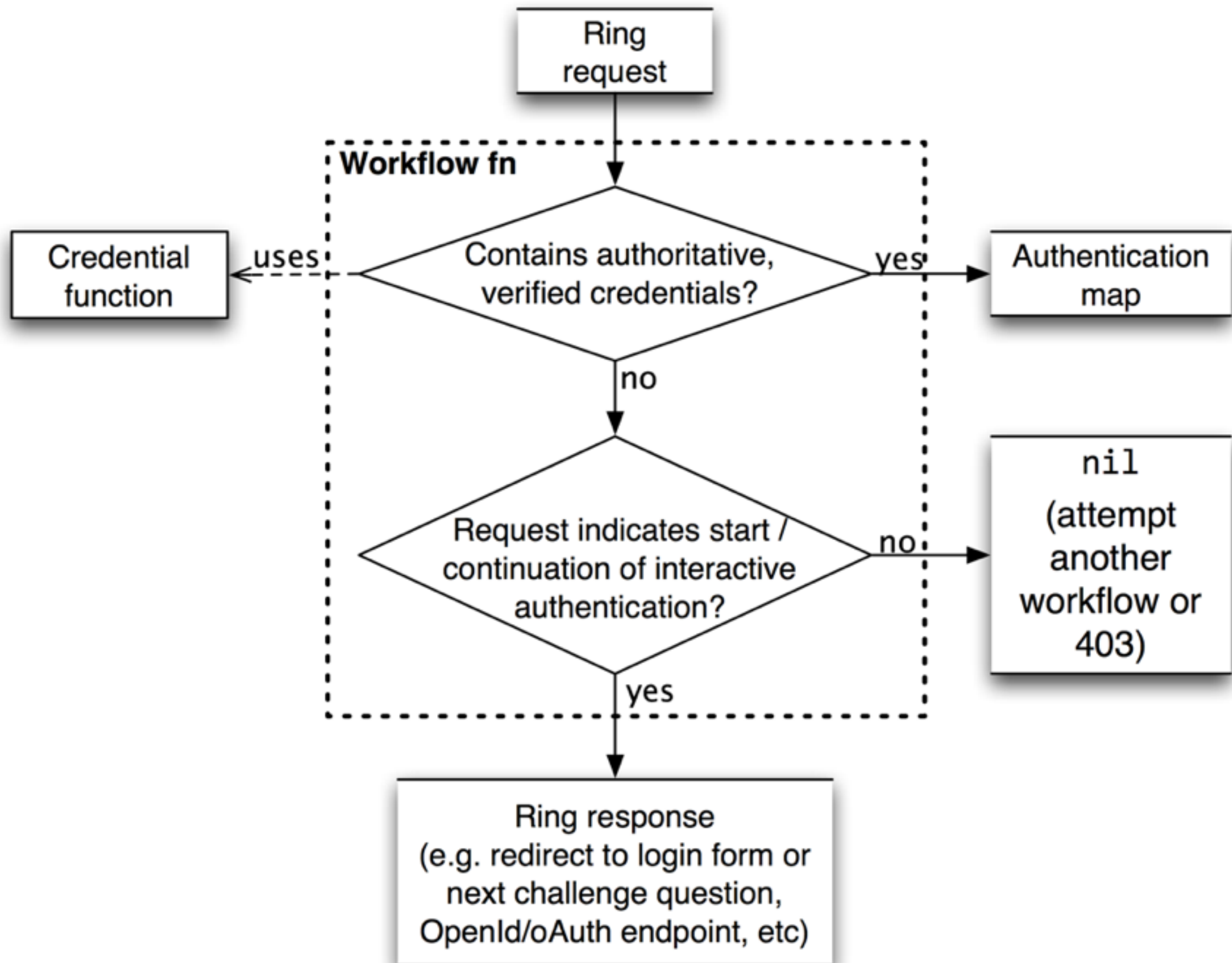
Batteries included

- Authentication workflows
 - form-based “interactive” login
 - HTTP Basic (Digest, soon)
 - OpenId (Google Yahoo Wordpress &c)
- “Channel enforcement” (e.g. require HTTPS)

Batteries included (2/2)

- Authorization options
 - Roles based on Clojure's ad-hoc hierarchies
 - Functions, macros, and Ring middleware for easily enforcing role-based authorization policy
 - Arbitrary imperative control

Architecture & Examples




```

(ns your.ring.app
  (:require [cemerick.friend :as friend]
             (cemerick.friend [workflows :as workflows]
                              [credentials :as creds])))

; assemble your ring app however you like
(def ring-app #_...)

; assert relationships between roles in your app (optional)
(derive ::student ::user)
(derive ::admin ::student)

; a dummy in-memory user "database"
(def users {"root" {:username "root"
                    :password (creds/hash-bcrypt "admin_password")
                    :roles #{::admin}}
            "jane" {:username "jane"
                    :password (creds/hash-bcrypt "Jane's password")
                    :roles #{::student}
                    :year 1}})

(def secured-app
  (friend/authenticate ring-app
    {:credential-fn (partial creds/bcrypt-credential-fn users)
     :workflows [(workflows/interactive-form)
                 (workflows/http-basic :realm "Friend demo") #_...]}))

```

:: enforcing arbitrary & role-based authorization policies

```
(compojure.core/defroutes app
  (GET "/requires-authentication" req
    ; scope within which a user must be authenticated in any capacity
    (friend/authenticated #_...))

  (GET "/course-schedules" req
    ; scope within which a user must have >= ::student role
    (friend/authorize #{::student} #_...))

  (GET "/admissions" req
    ; scope within which a user must have >= ::admin role
    (friend/authorize #{::admin} #_...))

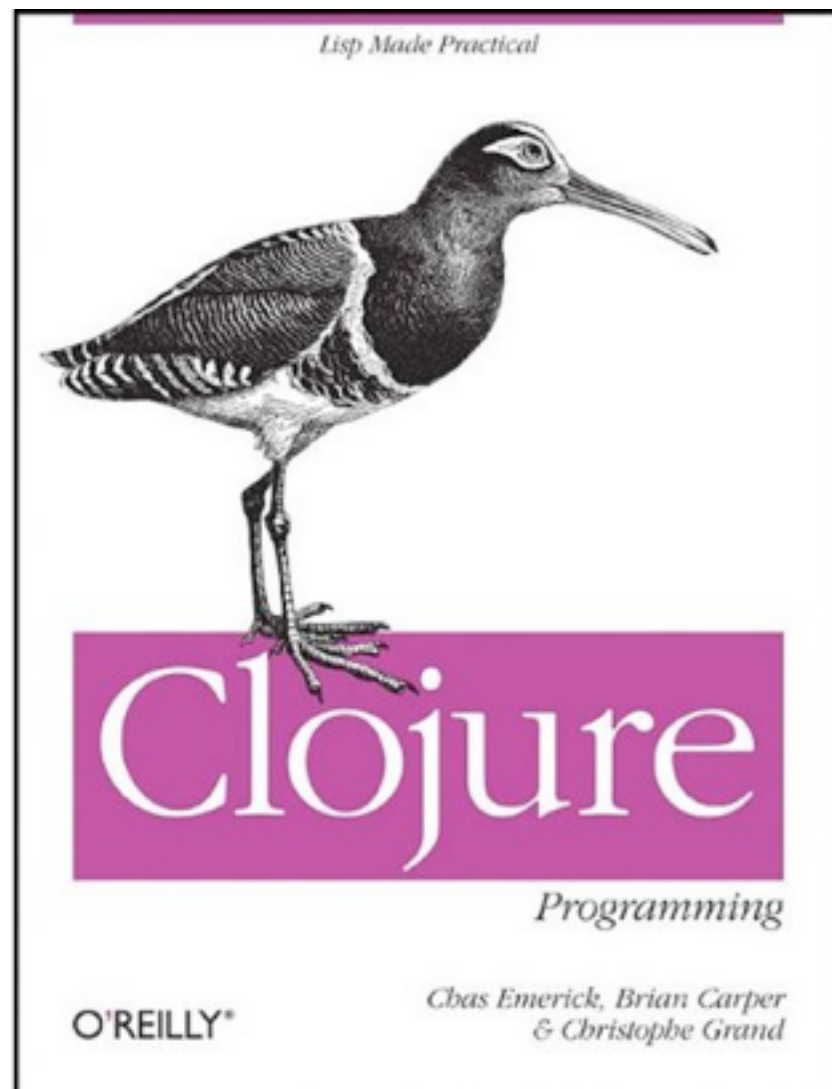
  (GET "/freshman-orientation" req
    ; arbitrary authorization criteria
    ; e.g. require that a user have >= ::admin role, OR be a freshman
    (let [id (friend/identity req)]
      (if (or (friend/authorized? #{::admin} id)
              (and (friend/authorized? #{::student} id)
                    (-> id friend/current-authorization :year (= 0))))
        #_...
        (friend/throw-unauthorized id
          {:reason "Must be a freshman to access orientation info"}))))))
```

Demo

Coming soon

- Factor out OpenID
- Canned support for OAuth2 providers
- Simplification of workflow contract
- More eyes / audit?

Questions?



<http://clojurebook.com>
[@ClojureBook](#)



<http://cemerick.com>
[@cemerick](#)