Security 2.0

How LinkedIn changed its security model in order to offer an API

Yan Pujante
Distinguished Software Engineer
Member of the Founding Team @ LinkedIn
ypujante@linkedin.com
http://www.linkedin.com/in/yan
Background

- LinkedIn created in 03/2003
  - close to 30M members
  - profitable with 5 revenue lines
  - HQ in Mountain View (400 employees)

- Technologies
  - 2 datacenters (~ 600 machines)
  - SOA, java, tomcat, spring framework, http, Oracle, MySQL, servlets, jsp, graph, OSGi...
  - dev on MacOS X, prod on Solaris
Content

- Part I:
  - Security model and how we changed it
- Part II:
  - API security: Yahoo vs Amazon models
- Part III:
  - Beyond API: scaling the website
Content

- **Part I:**
  - Security model and how we changed it

- **Part II:**
  - API security: Yahoo vs Amazon models

- **Part III:**
  - Beyond API: scaling the website
Simple model: BL checks permission level

- Guest ?
- Identified (remembered but not logged in) ?
- Logged in ?

If not => logging screen
### API / Partner

- Opening LinkedIn to partners: make calls on behalf of a user (REST api)
- Requires LinkedIn member to agree (one time operation)
Does not work with partner...

- Identity is different (partner with member)
- Partner levels (trusted, basic...)
- Different set of permissions
Model Change: from...

```java
public Profile getProfile(int memberId) throws UnauthorizedException {
    ensureIdentified();

    Profile profile = doFetchProfile(memberId);
    // ...
    return profile;
}

protected void ensureIdentified() throws UnauthorizedException {
    PermissionLevel level = getPermissionLevel(); // from thread local
    if (level == PermissionLevel.guest)
        throw new UnauthorizedException();
}
```
Model Change: to...

```java
public Profile getProfile(int memberId) throws UnauthorizedException {
    checkPermission(new Permission("readProfile"));
    Profile profile = doFetchProfile(memberId);
    // ...
    return profile;
}

private void checkPermission(Permission permission) throws UnauthorizedException {
    Subject subject = getSubject(); // from thread local
    Set<Principal> principals = subject.get Principals();
    for(Principal principal : principals)
    {
        if (principal.restricts(permission))
        {
            // one of the principals restricts the call
            throw new RestrictedException();
        }
    }
    for(Principal principal : principals)
    {
        if (principal.allows(permission))
        {
            // one principal is ok with the call
            return;
        }
    }
    throw new NoPermissionException();
}
```
## Principals / Permission

<table>
<thead>
<tr>
<th></th>
<th>Guest</th>
<th>Identified</th>
<th>Logged In</th>
<th>Trusted Partner</th>
<th>Basic Partner</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>read Profile</strong></td>
<td>❌</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td><strong>add Connection</strong></td>
<td>❌</td>
<td>✔️</td>
<td>❌</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td><strong>close Account</strong></td>
<td>❌</td>
<td>✔️</td>
<td>❌</td>
<td>✔️</td>
<td>✔️</td>
</tr>
</tbody>
</table>
RBAC stands for: Role-Based Access Control

- Subject has 1 or more roles (principals)
- Permissions are assigned to specific roles

➤ You don’t do:
  - are you in role R ? (are you Identified ?)

➤ Instead you do:
  - are you in a role that gives you permission to execute operation O ? (checkPermission)
Content

- Part I:
  ➢ Security model and how we changed it
- Part II:
  ➢ API security: Yahoo vs Amazon models
- Part III:
  ➢ Beyond API: scaling the website
API security / Yahoo

- Yahoo model
  - login => get back sessionID
  - all requests include sessionID (expires)
  - client must manage sessionID

- To be user friendly code must do:

```python
try
    execute request
catch error
    if error == session expired
        login
        execute request
```
API Security / Amazon

- Amazon model
  - Easier to implement
    - no more login, all requests are identical
    - no recovery to implement
    - no session to manage
  - Safer
    - valid a shorter amount of time
    - sign other parts of the request
Authorization Header

- Each request has a new header
  authenticate + signs the request

PTM:<ptrID>/<memberKey>:<now>:<signature>

- ptrID = partner ID
- memberKey = member agreement
- now = current time in seconds
- signature = signs this header + request
Signature

Hexa(HMAC-SHA1(UTF-8-Encoding-Of(StringToSign)))

StringToSign = "PTM\n" +
   Entity + "\n" +
   now + "\n" +
   HTTP-Method + "\n" +
   Request-URI + "\n" +
   Content-Type + "\n" +
   Content-Length + "\n" +
   Content-MD5 + "\n"

With:
Entity = <api_key>/<agreement_key>
now = number of seconds since Epoch (01/01/1970 UTC midnight)
HTTP-Method = "GET" | "POST"
Request-URI = everything after the host:port part of the URI
Content-Type = content of the header ("" if header missing)
Content-Length = content of the header ("0" if header missing)
Content-MD5 = content of the header ("" if header missing)
Example: POST

Example:

- ptrID = abcdef
- memberKey = 1234
- Shared Key = “9876543210”

```
POST /foo HTTP/1.1
Content-Type: text/plain
Content-Length: 100
Authorization: LINAPI PTM:abcdef/1234:1189532154546:786533ec913320fd31ebaee878f055cf
```

String To Sign:

```
PTM
abcdef/1234
1189532154546
POST
/foo
text/plain
100

```
API Security / Amazon

- Drawback
  - bigger samples => easier to ‘guess’ the shared key
  - does not prevent re-playability issue

- Solution
  - 2 shared keys
    - one for HTTP calls
    - one for HTTPS calls
  - API is read-only for now... need to fix it
Example: python

```python
import time
import hmac
import sha

def build_header(requestURI, memberKey):
    nonSecureSharedKey = "xxxxxx"
    secureSharedKey = "sssss"
    api_key = "2000_ACME"
    now = time.time()
    entity = api_key + '/' + memberKey
    stringToSign = "PTM\n"
    entity + "\n" + \
    str(int(now)) + "\n" + \
    "GET" + "\n" + \
    requestURI + "\n" + \
    "\n" + \
    "\n"
    sig = hmac.new(nonSecureSharedKey, stringToSign, sha).hexdigest()
    return {
        'Authorization': 'LINAPI PTM:' + entity + ':' + str(int(now)) + ':' + sig
    }
```
Content

- Part I:
  - Security model and how we changed it
- Part II:
  - API security: Yahoo vs Amazon models
- Part III:
  - Beyond API: scaling the website
The website

- Ported the concepts to the website

  LIM:<memberID>;<level>;<now>;<signature>

- Signature uses same algorithm
  - signs the rest of the cookie
  - uses shared key (and secure shared key)

- Every request ‘recovers’ the Subject:
  - verify signature
  - check expiration
New Architecture

- Same path:
  1. recover Subject
  2. execute BL call
---

**Single Sign On**

- Separate server for single sign on
- Stateless front-end (good for scalability)
Conclusion

- Pitfalls
  - permission representation is a nightmare: big + version
  - communication between servers makes it hard to track problems

- Good news
  - It works!
  - It is scalable and flexible