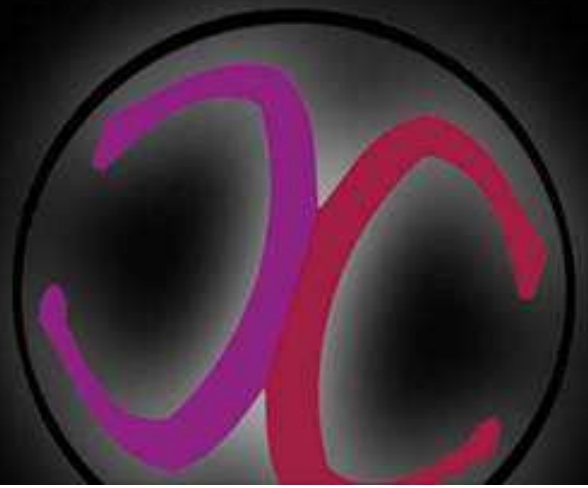
RETS Transport Authentication

April 2014

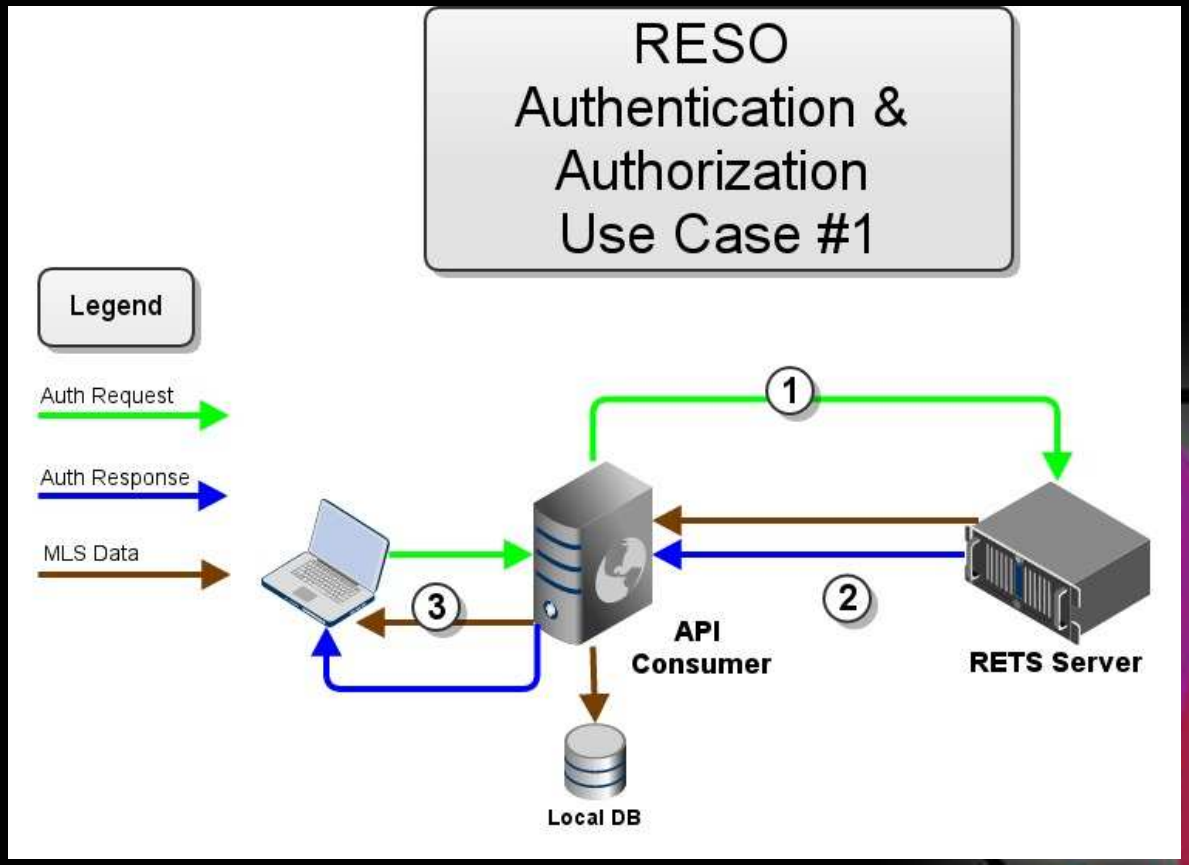
Matt Cohen
Clareity Consulting
Clareity.com



Use Cases (Thanks Cal!)

1. **SP (Service Provider) to SP/IdP (Identity Provider) - Server or Client to Server authorization without human intervention**

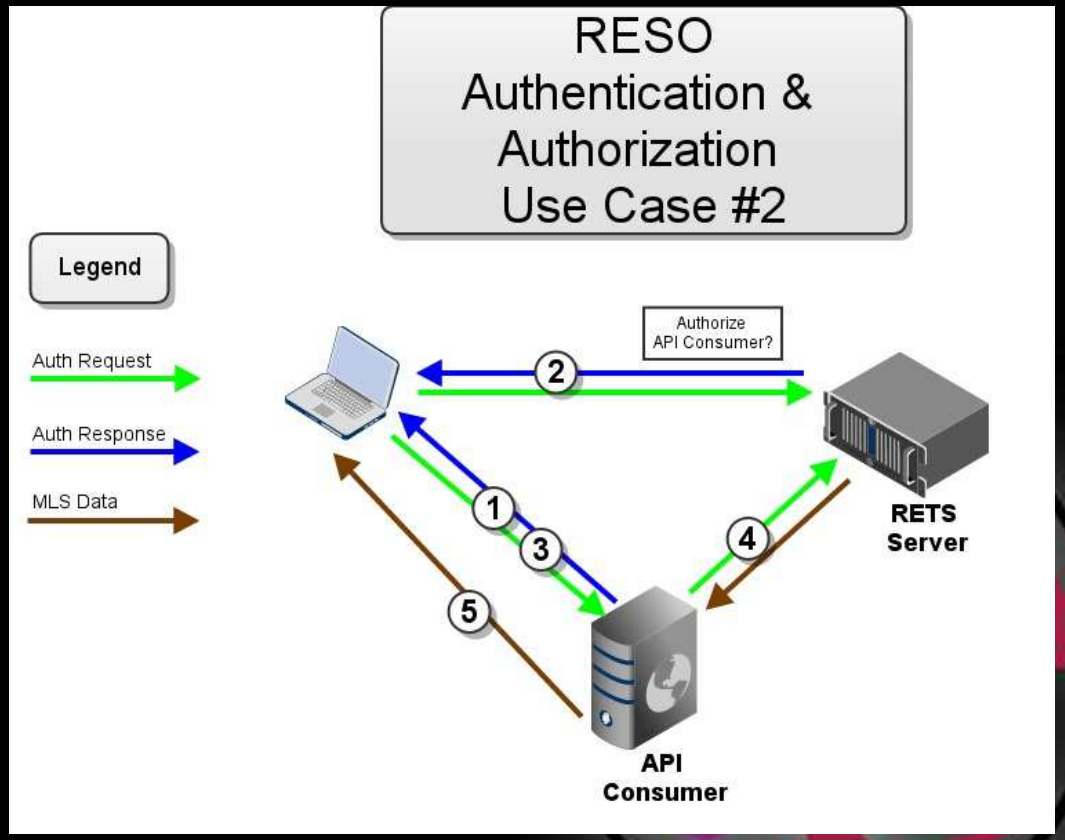
Example: RETS 1.x style flow. A syndicator's recurring bulk download of listing data.



Use Cases

2. SP to IdP to SP: Typical three-way authorization of a user. (Transient authentication of an API Consumer on behalf of an MLS member)

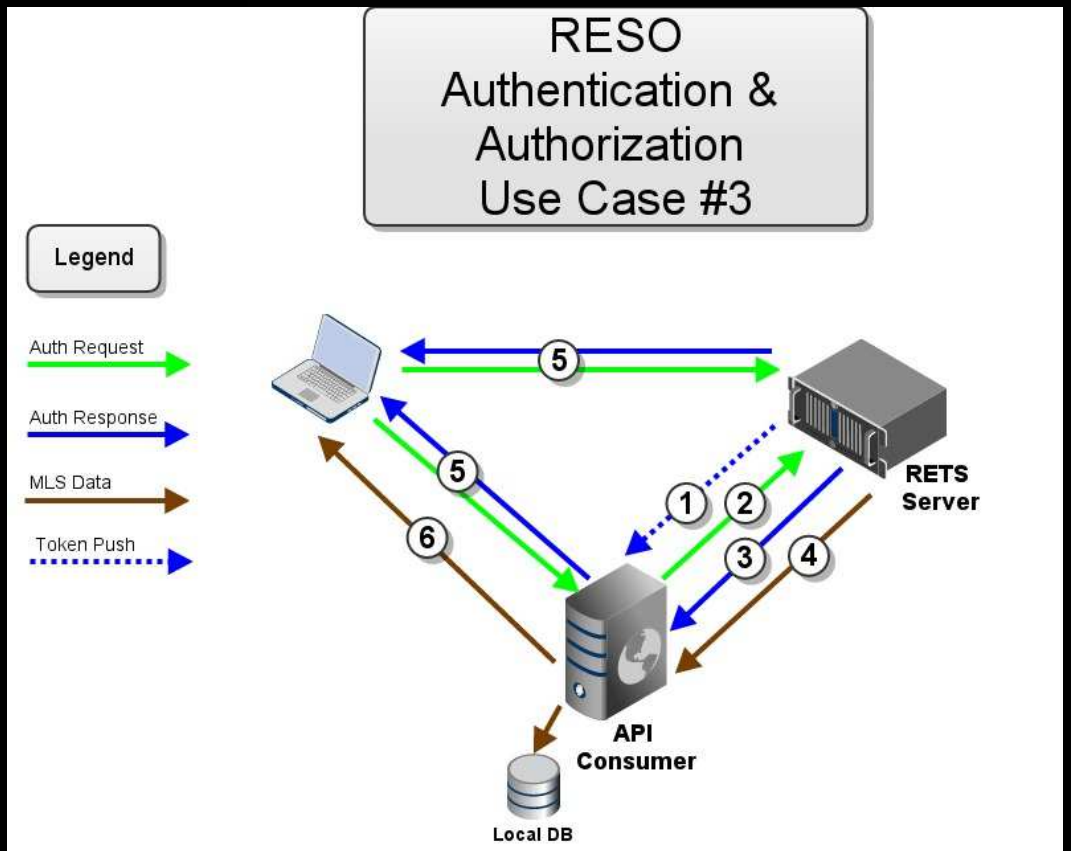
Example: A web application that interacts with the MLS on behalf of a user, e.g., a real-time CMA.



Use Cases

3. SP to SP/IdP: Transparent three-way authorization of a user. (Transient authentication of an API consumer on behalf of a user without human intervention)

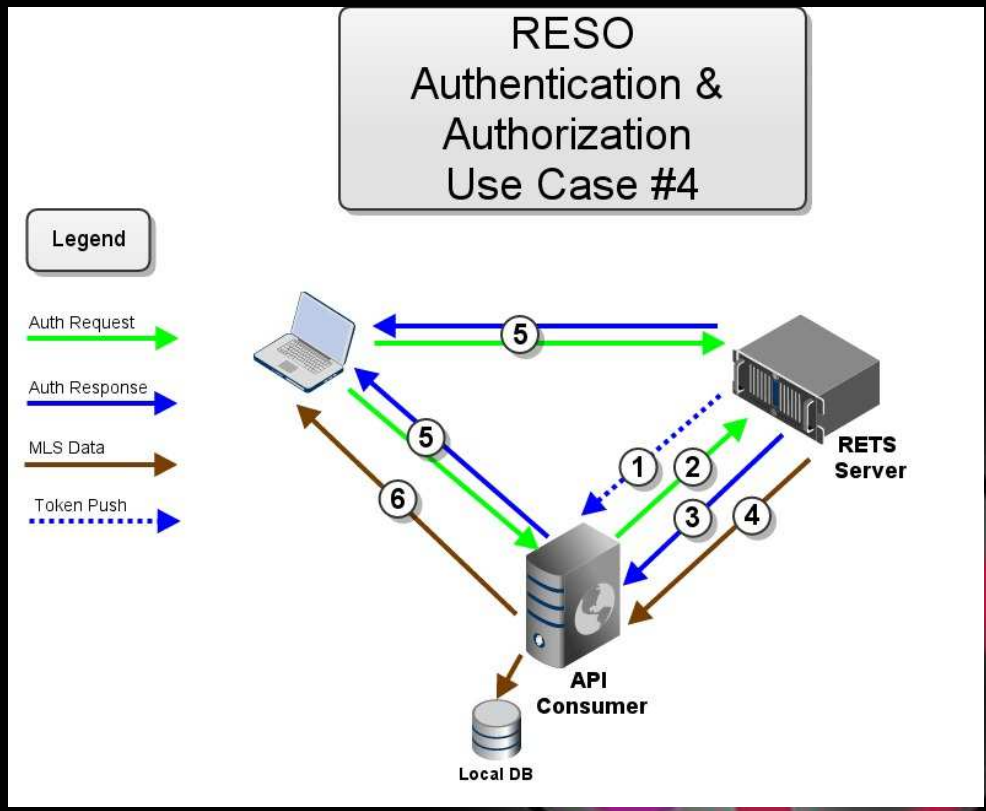
Example: A VOW provider's validation of eligibility for an existing customer.



Use Cases

4. SP to SP/IdP: Transparent, recurring "on behalf of" authorization of a user. (Persistent, transient authentication of an API consumer on behalf of a user without human intervention)

Example: Lead Management software that pulls leads from multiple sources for a given customer.



Standards Recommendations

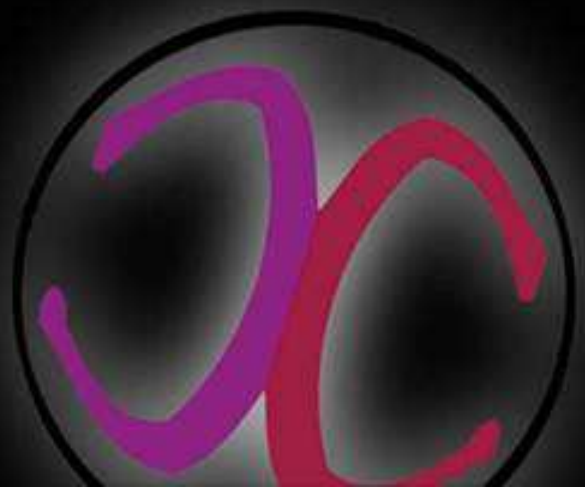
Following are the recommended standards and their uses:

- **HTTP Digest Authentication** SHOULD be supported, as the easiest standard to implement which addresses the first and most prevalent use case for RETS, and which can be made to serve some other use cases as well.
- **oAuth 2** SHOULD be supported as needed to support additional use cases, especially where three-legged authorization is required.
- **SAML 2.0 Bearer Assertion Grant Type Profile for OAuth 2.0** In environments where SAML is already in use, SAML MAY be used as an oAuth Profile.
- **SAML**. In environments where SAML is already in use, SAML MAY be used.

Beyond

The final document includes:

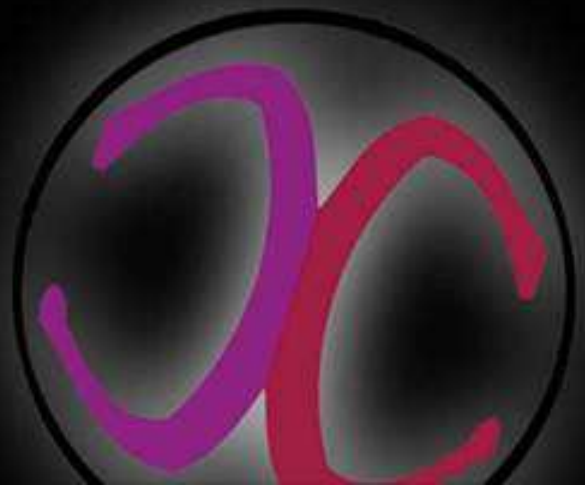
- Implementation recommendations
- Code examples
- Links to toolkits for each standard



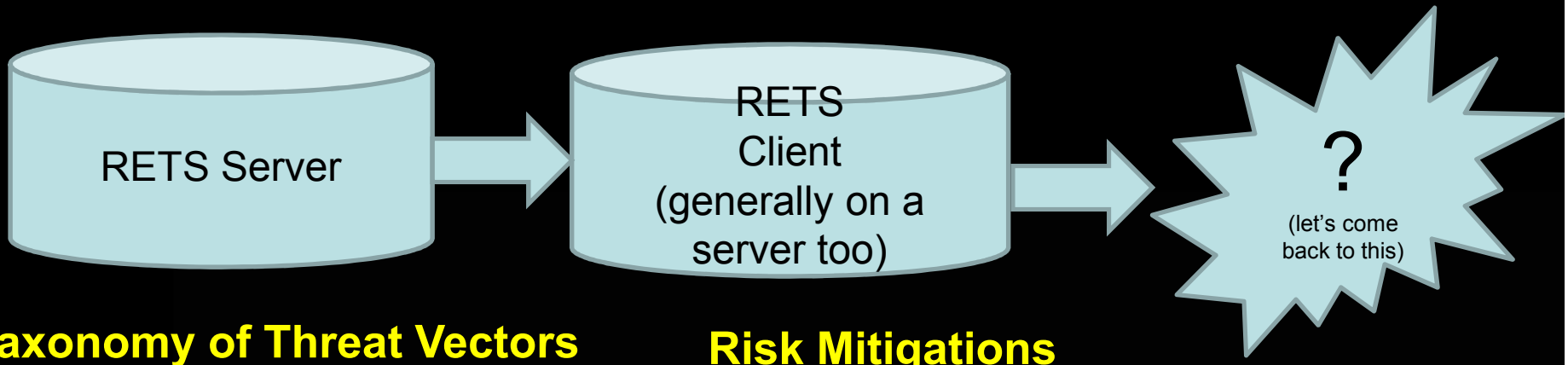
API (RETS) Security

April 2014

Matt Cohen
Clareity Consulting
Clareity.com



RETS Security Today



Taxonomy of Threat Vectors

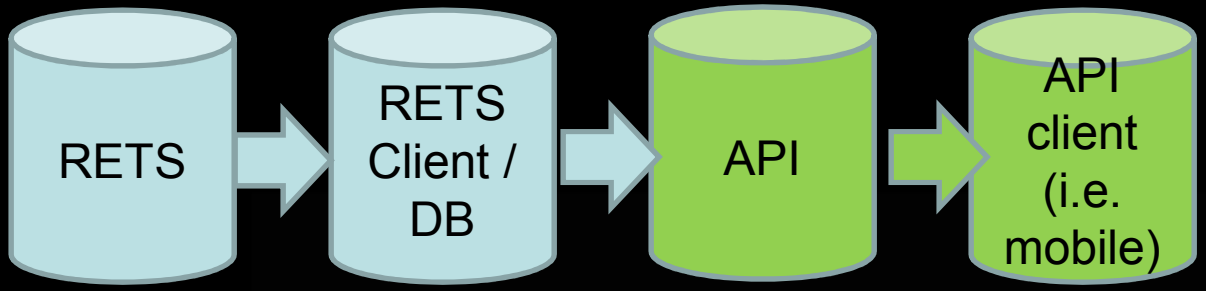
1. Hacking
   a. Server
   b. Client
2. Credential Sharing
3. Credential Theft (in transit)
4. Session Hijacking / Data Theft
5. *Illicit Data Sharing (after the transmission)*
6. *Data misuse (after the transmission)*

Risk Mitigations

1. Since both are server based, high amount of protection possible. Policy, Physical, Personnel, OS, Software. When client *not* on a server – storage encryption w/ key protection.
2. IP Address Restrictions, Usage Tracking -> Credential Revocation, password policies
3. #2 + Digest Auth, SSL (rare!)
4. SSL (rare!)

5-6 ??? – uses are often derivative products or dark web – legal recourse.

API Security (NOT just RETS)



What I've Seen in the Field – API Step and Beyond

1. APIs (restful and otherwise) with *no* credentials
   - Surprisingly common!
2. Credential Sharing / Theft (all steps)
   - Very little tracking / remediation
3. Session Hijacking / Data Theft (all steps)
   - Use of encryption (even RETS supported) not common
4. *Data Sharing / Misuse*
   - Individual legal action ineffective. Industry just starting to organize (red-plan.org)

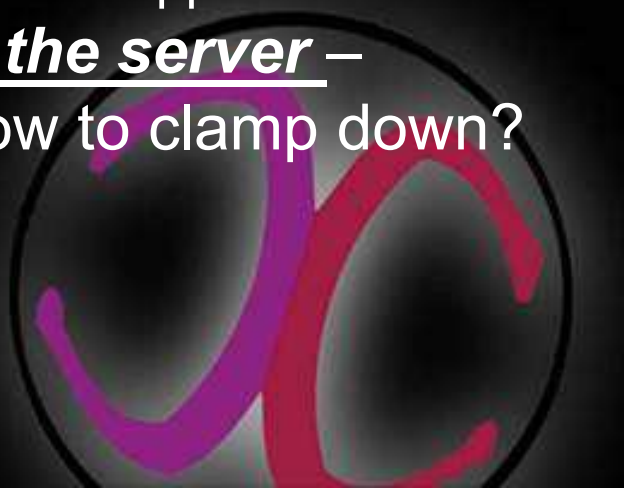
Clareity

Mobile API (inc. NEW RETS) Special Challenges

- **Server-side -> Server client use is *less* an issue.**
- ***Can* be limited by IP and/or tracked by IP**

Mobile? Especially using single "app" credential to the API?
Credentials then can be misused – for another app entirely
Yes, this isn't just theoretical – it's already happened.

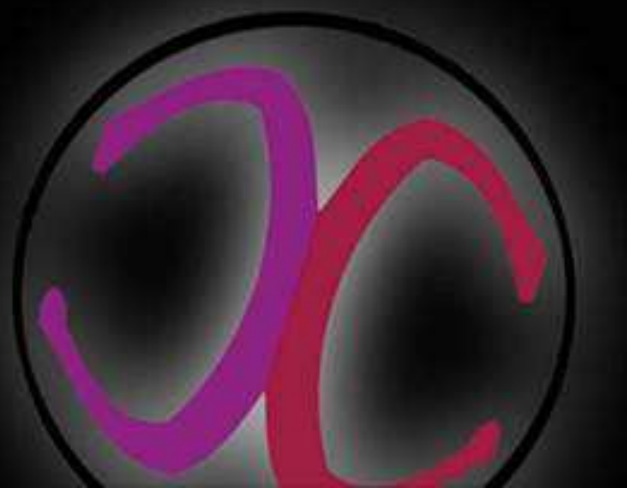
- Credentials (and data stores) highly vulnerable to misuse
  - Client credentials can be taken from disassembled app.
  - Client credentials packet sniffed, easy 'local' to the app.
- ***One app looks just like another app to the server*** – especially if *generic credentials* used – how to clamp down?



Mobile API (inc. NEW RETS) Special Challenges

Now we want to release mobile-friendly RETS!

- How much protection is enough? (Imagine I'm auditing on behalf of an MLS…)
- Goal for RESO: best practices doc (can be used to create development contracts?)



Clareity

Let's Dig In - Restful API

- **Credential-less restful API looks like a web address – usually easy to traverse**
  - http://rest.DOMAIN.com/search/... /[search parameters]
  - http://rest.DOMAIN.com/listing/... /4439594
- **Today: often depending on "obscurity" for security – hopefully we all agree – not good enough.**

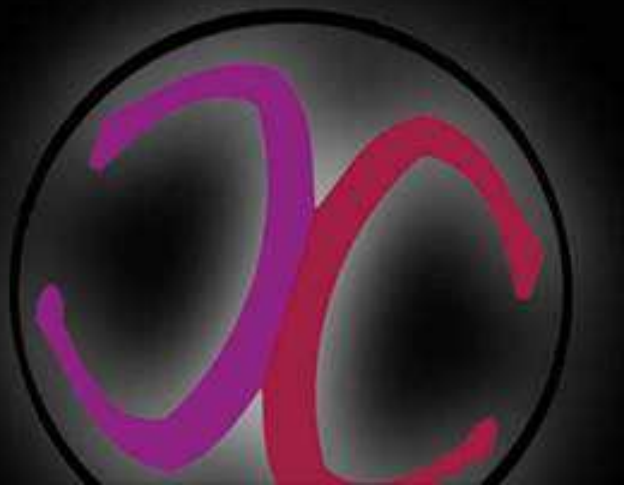
- "**Anti-scraping" / security measures?** No …
  - Pages (listings) / minute rate limiting?
    - Nope - slow crawl, multi-credential, multi-IP use. That alone isn't enough.
  - Pages / "session"? Session length?
    - What's a session in the restful API context? (Also, as above)
  - It's *ALL* 'bots'!!!

Clareity

Add individual credentials?

http://rest.DOMAIN.com/listing/... /4439594

- **Require creation of *individual* credentials?** Can track credential overuse patterns.
  - **BUT …** Data thieves can register / use multiple credentials to defeat overuse protection.
    - total listings / typical pages viewed per day = # of IPs / users to create.
    - Cloud makes it easy to spin up unique IP servers.
- Maybe this is *part* of a solution?



Clareity

Matt's Non-Ideal Solution

- Generate new API passwords every [short] period – make them difficult to reverse engineer.
  - Put a regularly changed key in difficult to-directly-access protected storage (*is anything protected enough on Android or IOS?*)
  - Combine with part of a GMT timestamp
  - Combine with other information provided to the app from the server or vice versa (i.e. a checksum on app size for current version)?
  - Encrypt *before* transmission
- Server-side, a matching hash would be made to check against - allowing the previous one to work for a short grace period, of course).

Mike Sparr: Abandon restful

- Manage sessions are handled server side.
  - Can monitor session patterns
  - Generates authentication tokens with hybrid of:
    - nonce
    - identifier (UUID-like)
    - and other factors (mix of client and server side content so someone decompiling the app cannot easily figure it out)
  - Regular session expiration requiring re-authentication adds additional security.

Similar to my approach. But session oriented.

Mark Flavin: use client side certificates

- Generate and store client side certificates and use those for ongoing logins this would eliminate the need to cache credentials locally and would create a more secure authentication which remains flexible.
  i.e.
  - http://www.techrepublic.com/blog/software-engineer/use-https-certificate-handling-to-protect-your-ios-app/
  - http://chariotsolutions.com/blog/post/https-with-client-certificates-on/

- Why can't hackers use the client side certificates?
- Isn't this just another credential that can be misused?
  - How does the server know an app's okay to request the certificate?
- It does increase effort (but on both sides?)

Cal Heldenbrand says…

- Use unique IDs *and add …*
  - Email address with verification.
  - Pair ID with an access token, or maybe a completely unique client_id/secret.
  - Identities could still hypothetically be generated in mass quantities, but it's still a difficult task, and it's easier to track server side. If you usually see about 5 registrations per week, and all of a sudden you see 1,000 in an hour, then you block that IP for a while. Or maybe just invalidate that huge batch of access tokens that were generated from that one IP. You could also just rate limit the token generation endpoint. And once the access tokens are unique for each identity, you rate limit the access token.

Clareity

Discussion

What we're really talking about is whether there should be best practices (minimum standards?) for:

A. Establishing *individual* access tokens (to use as credentials in oAuth2, Digest, etc.)
B. Making token more difficult to access – even with device / source code / network access
    - Matt&Mike: Generate new API passwords every short period – make them difficult to reverse engineer?
    - Mark: Certificates?
C. Standards for issue identification (logging, pattern recognition, alerting) & remediation
    - Cal: watch for unusual patterns of credential creation
    - Matt: watch unusual usage patterns? (easy to game)
    - If we publish standard recommendations won't someone just take advantage and 'game' them?

Thank you!

Matt Cohen
Matt.Cohen@Clareity.com
Clareity Consulting
Clareity.com