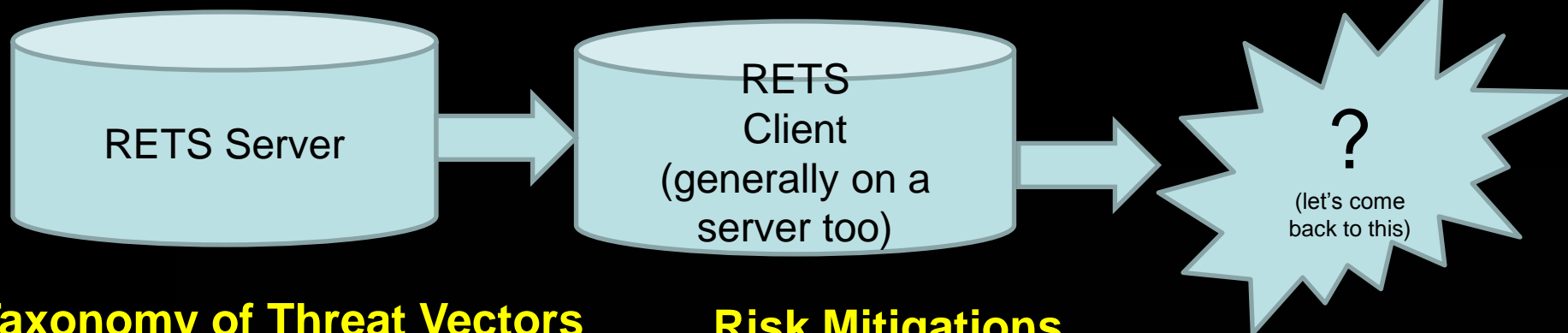


API (RETS) Security

April 2014

Matt Cohen
Clareity Consulting
Clareity.com



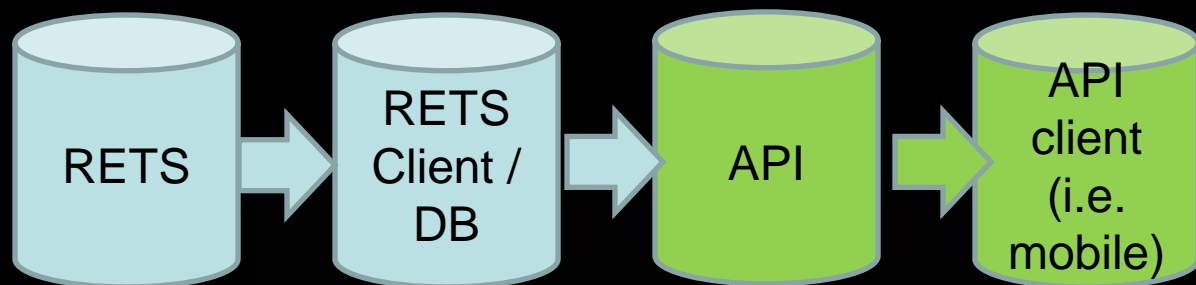


Taxonomy of Threat Vectors

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 (sadly rare!)
4. SSL (sadly rare!)
- 5-6 ??? – *uses are often derivative products or dark web – legal recourse.*



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)

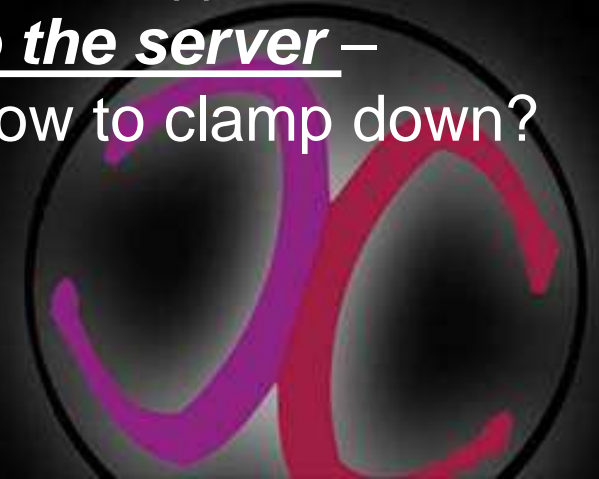


- Server-side -> Server client use is *less* an issue.
- **Can be limited by IP and/or tracked by IP**

**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?



Now we want to release mobile-friendly RETS!

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



- **Worst case seen in the field (using YOUR DATA): credential-less restful API looks like a web address – usually easy to traverse (I'm not talking RETS here)**
 - `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 ... It's *ALL* 'bots'!!!



[http://rest.DOMAIN.com/listing/... /4439594](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?




- 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).



- 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.



- 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?)
- 

- 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.

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?

Please Email Me Your Recommendations!

Matt Cohen

Matt.Cohen@Clareity.com

Clareity Consulting

Clareity.com

