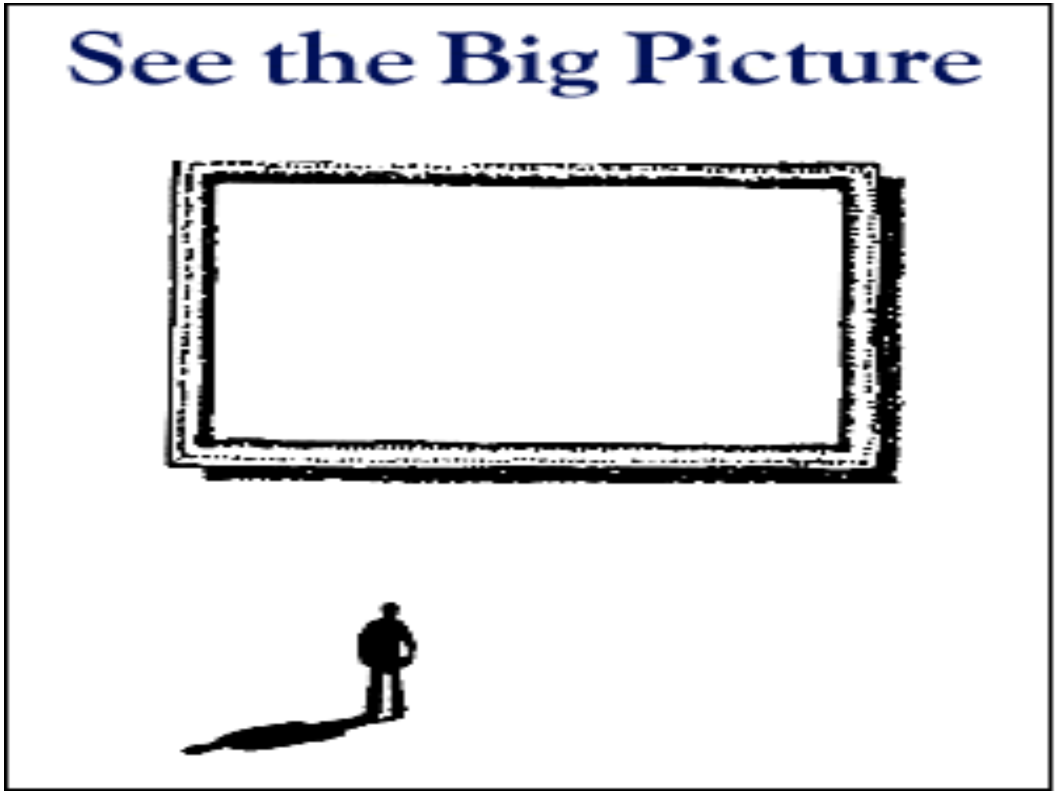
Life of a Password

Arvind Mani
Data & Infrastructure Security
LinkedIn



Secure user account.

Agenda

We cover the following topics:

- Hashing
- Transport
- Storage

Not covered:

monitoring, host and network security, access control, other account protection mechanisms.

Unsalted Password Hash

User id	hash
m1	F(“monkey”)
m2	F(“123456”)

- Brute force short passwords
- Dictionary attack
- Rainbow Table

Salted Password Hashes

Cheap Salt (performance; what’s wrong?)

User id	hash
m1	F(“monkey”, m1)
m2	F(“123456”, m2)

Random Salt

User id	hash	salt (64/96 bit)
m1	F(“JI6aerwhm”, s1)	s1
m2	F(“$^%YRTYFYU”, s2)	s2

Susceptible to Targeted attack

Keyed Crypto Hash (MAC)

User id	hash
m1	F("Jl6aerwhm", "**secret**")
m2	F("$^%YRTYFYU", "**secret**")

- Prevents dictionary attack
- Common passwords are revealed
- Prevents targeted attack

Next: Online attacks

Overwrite Attack

User id	hash	salt
m1	F("ashjrgqwu3nk", s1)	s2
m2	F("%RYThj#WY", s2)	s2

User id	hash	salt
m1	F("password", s)	s
m2	F("password", s)	s

Attacker overwrites m1 and m2's real passwords

Swap Attack

User id	hash	salt
m1	F("password", s1, "**secret**")	s1
m2	F("$^%YRTYFYU", s2, "**secret**")	s2

User id	hash	salt
m1 (attacker)	F("password", s1, "secret")	s1
m2 (victim)	F("password", s1, "secret"	s1

Keyed Hash

Pros

- One-way
- Correlated input secure

Cons

- Hash computation is fast
- Fixed input, fixed output

Password Recipe

- Key Derivation Function (KDF) instead of crypto hash
- Random salt
- User or member id
- Work factor (active accounts)
- Application secret
- Encrypted Hashes vs MAC

Ongoing Key Rotation

- Increase likelihood that not all stored credentials can be cracked.
- You have fingerprinted your database – stolen hashes can pinpoint “when”

Except if…

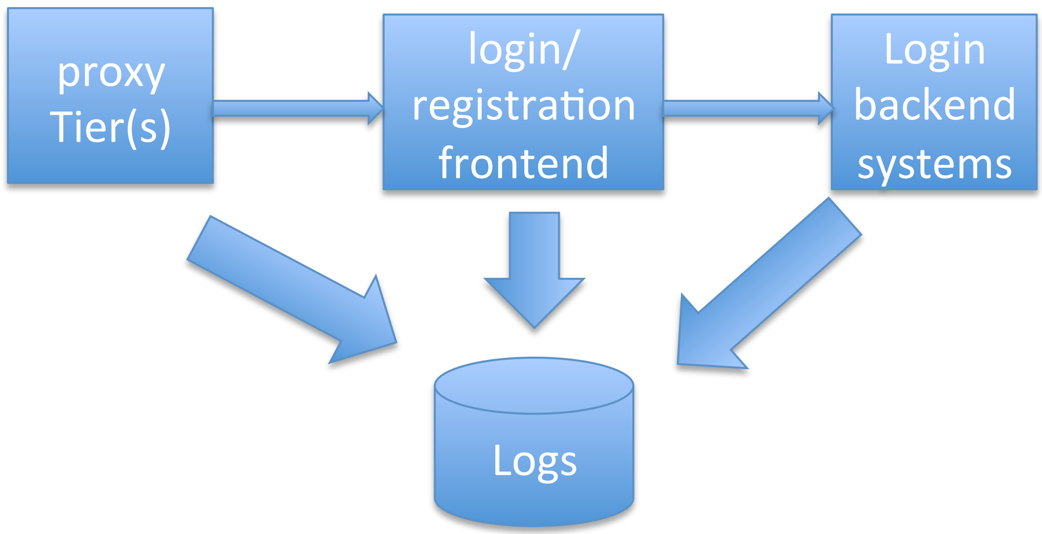
Password History Table

User id	hash	salt
m1	F("password", s1, m1, "secret")	s1

Password Table

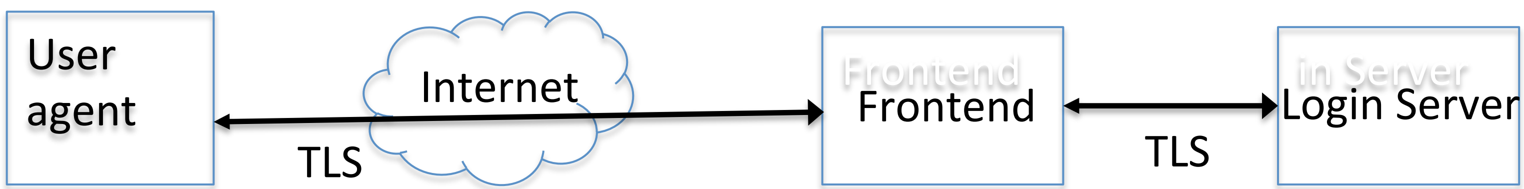
User id	hash	salt
m1	F("^%$TRsfwe", s2, m1, "secret")	s2

Accidental Logging



2014/02/25 18:38:55.751 [(prod-host1,login-app,/login,2014/02/25 18:38:55.572) verifyPassword(email="foobar@yahoo.com", **password=monkey**, ip_address="1.1.1.1"), PASS, 11ms

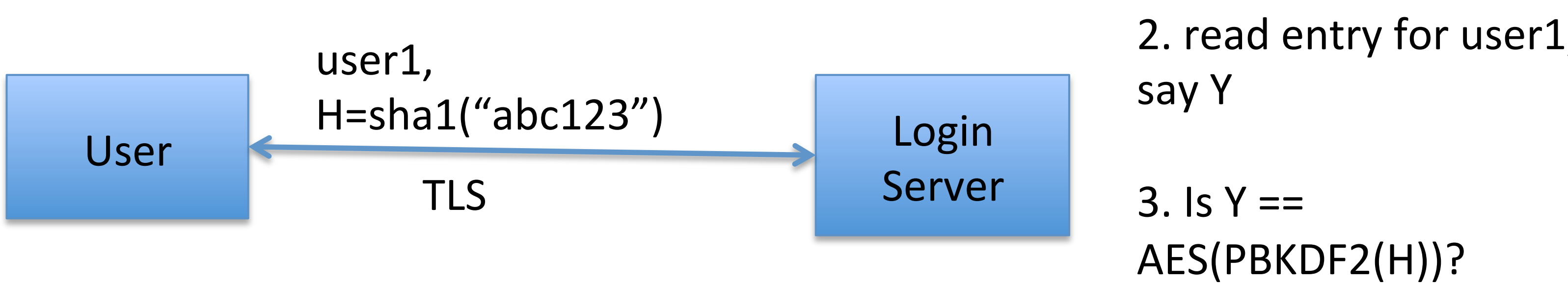
Transport



- TLS throughout, so on network password is always encrypted
- Each hop sees password in clear – potential for improper handling

Fix at User Agent – Attempt 1

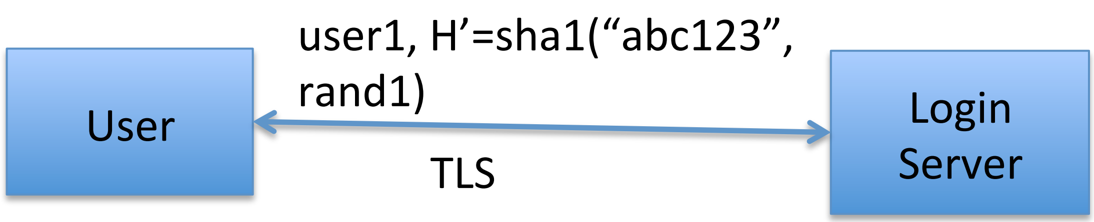
Send Hash(password)



- Equation in Step 3 holds if the sha1 is done consistently during registration, password reset, etc
- **Problem** – hashed password log is as bad as logging cleartext password!

Fix at User Agent – Attempt 2

Send Hash(password, salt)



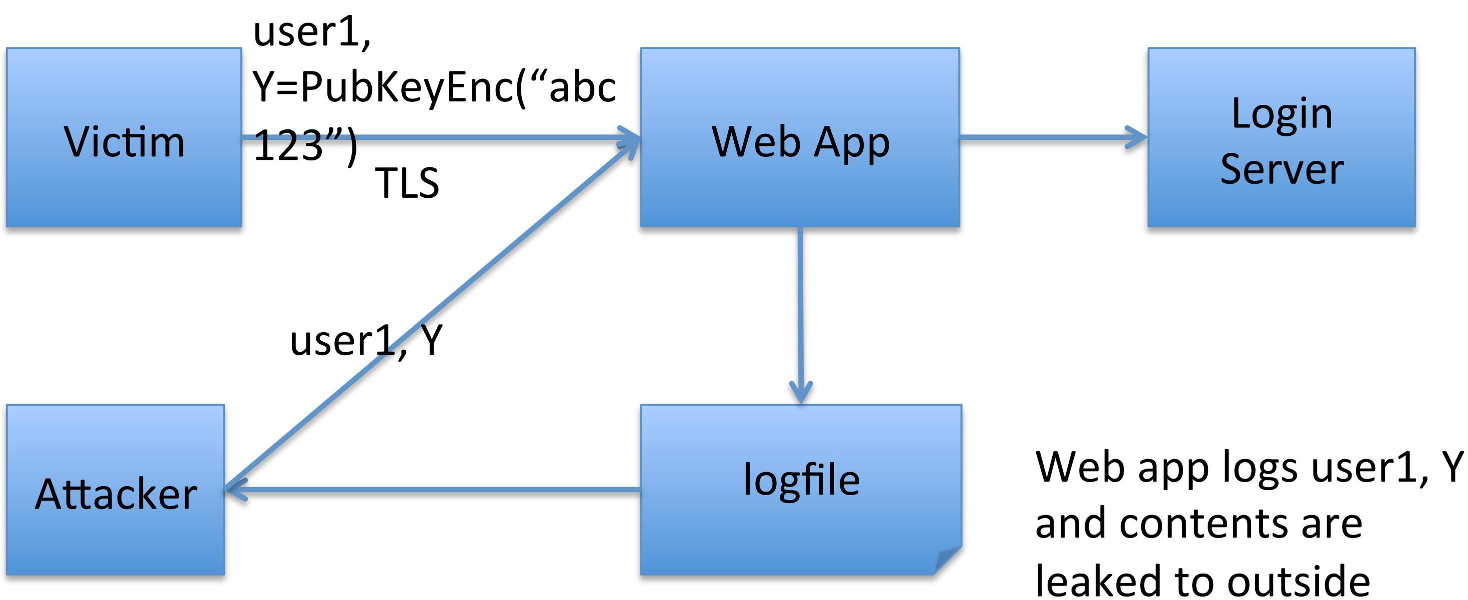
2. read entry for user1, say Y

3. Is Y == AES(PBKDF2(H'))?

- Salt used in computing Y using PBKDF2 can't be same as rand1
- **Problem** - Equation in Step 3 can't hold for any verification, scheme not feasible

Fix At User Agent – Attempt 3

Send PublicKeyEncryption(password)



Problem – Can replay and use encrypted password instead of real password to login as user

Fix At User Agent - Summary

Send PublicKeyEncryption(password) + nonce

- Good news – this finally works!
- Bad news – must support all user agents including native mobile, some clients can't be upgraded

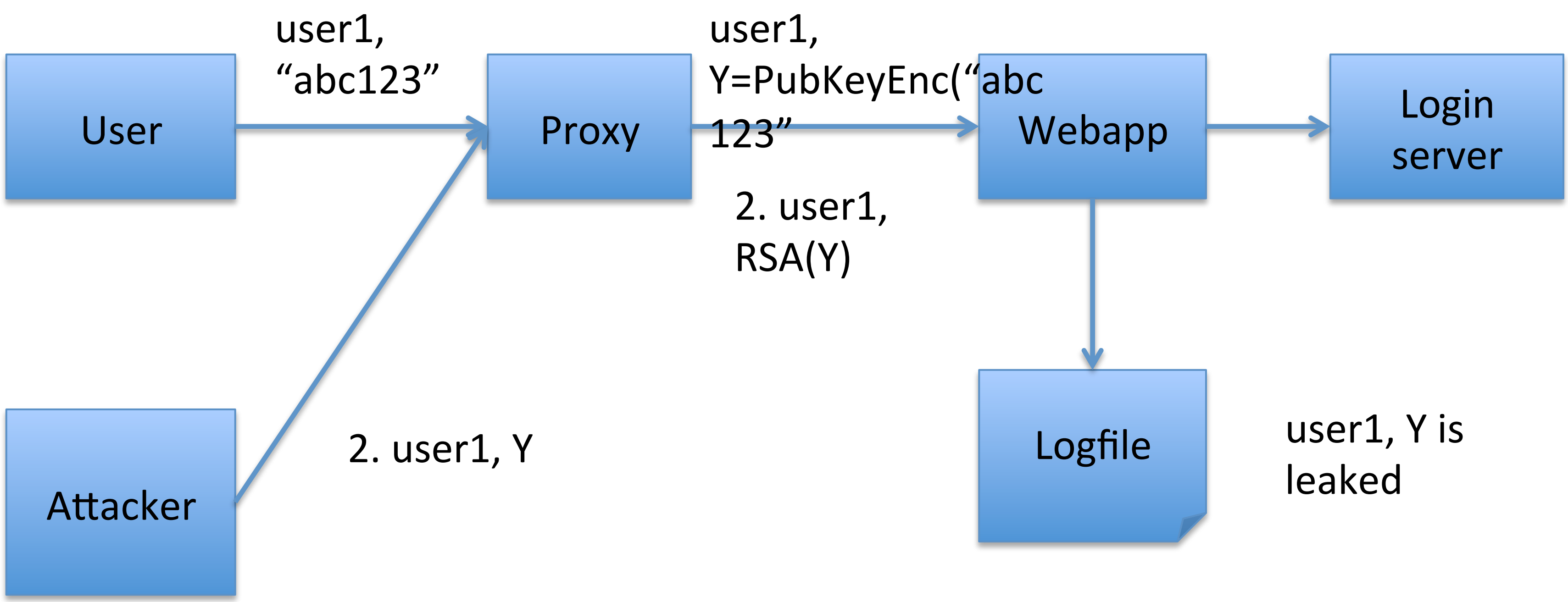
Fix at Ingress – Attempt 1

Instead of sending password, over TLS send either:

- Hash(password, salt)
- Password token not derived from password

Fix at Ingress – Attempt 2

PublicKeyEncryption(password)



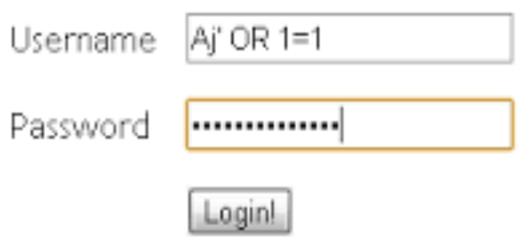
No replay from outside, can replay from inside network

Cloaked Password

- Password encrypted $PublicKey_{loginserver}$
- Ciphertext is randomized
- Replay protection via short expiry or nonce infrastructure
- Can be decrypted only by verification end point

Storage

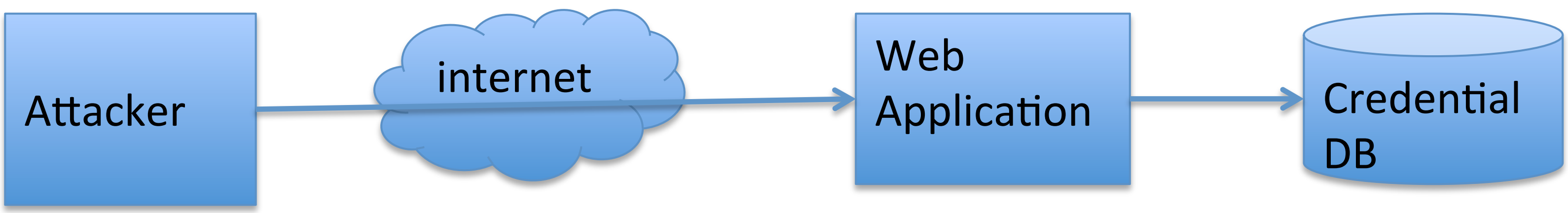
- SQL injection



- Attacker has username/password of database
- Attacker has access to filesystem

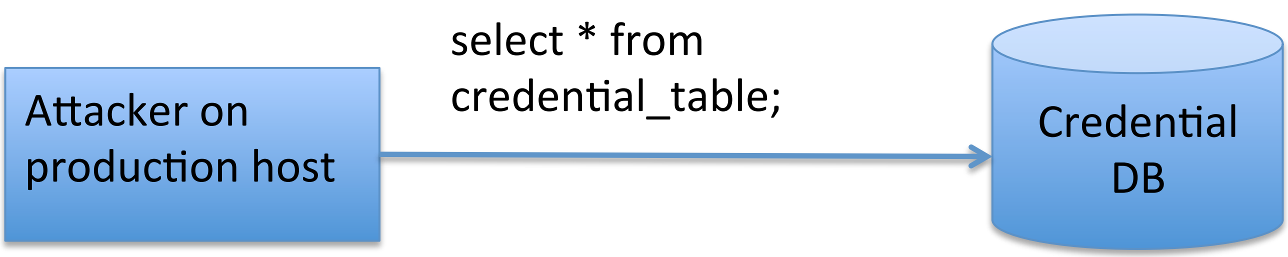
Dump credentials

- SQL injection (nosql stores are not by default safe)



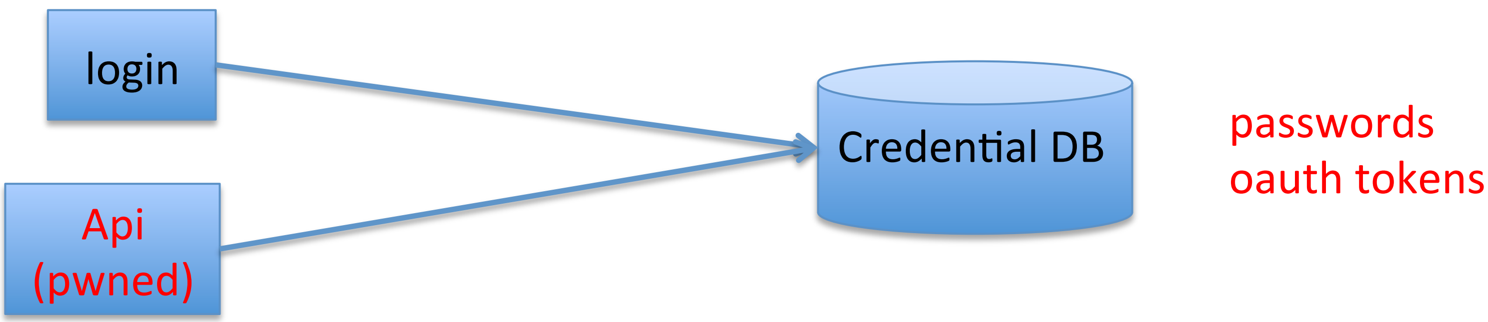
password='foo' or 1=1 --

- Attacker with DB credential

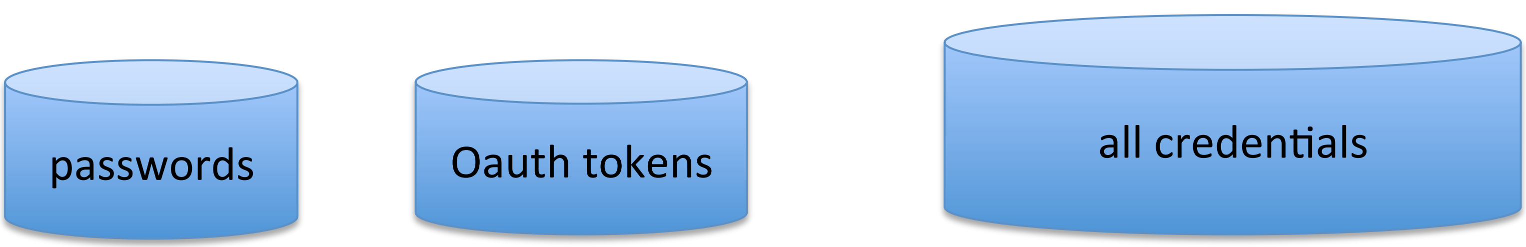


Centralizing Storage

- Many types of credentials – isolate application credentials



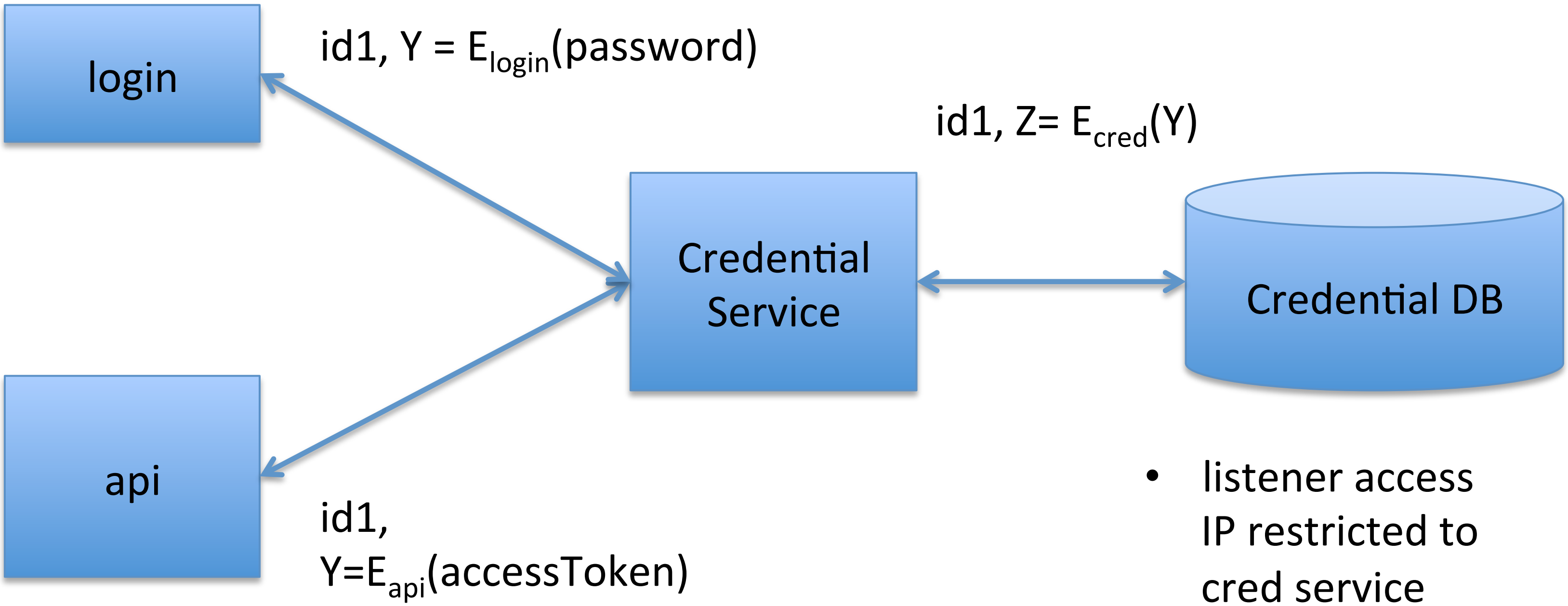
- Single point of attack



Credential Store

- Access via Stored Procedure
- Isolate client data via dual encryption
- Access Control
- Auditing
- Monitoring
- Periodic key rotation

Credential Store



- listener access IP restricted to cred service
- Access via Stored procs

- All communication over TLS
- ACLs on operations
- Client encryption

Summary

- Made some progress securing passwords
- Re-usable infrastructure – apply to credit cards, OAuth tokens, etc
- Future Work – Key Management, SRP?, mitigate risk of compromise of critical applications

Acknowledgement

We want to thank Professor Dan Boneh, Applied Crypto Group, Stanford University for his help with password hashing scheme.

Questions?

amani@linkedin.com

https://www.linkedin.com/in/arvindmani