Login Protocols

Rohit Musti

CUNY - Hunter College

May 1, 2022

Motivating Examples

- Different systems demand different kinds of security!
- If you were trying to unlock your phone or a local computer, you can think of this like unlocking a door, a key should suffice
- If you were trying to unlock a car using a key fob, then we need to design that protocol s.t. any adversary cannot replicate the radio waves
- If you are trying to pay for groceries and someone tries to steal your card information, an ideal protocol wouldn't allow someone to reuse the interaction
- If you are the victim of a phishing attack, ideally someone couldn't reuse the credentials to access your logins!

Attack Modes

- **Direct Attacks:** only publicly available parameters!
- **Eavesdropper Attacks:** listening in on the transcriptions of interactions
- **Active Attacks:** actively providing malicious interactions

Key Types

- **Secret Keys:** the verifier keeps their verification key private
- **Public Keys:** the verifier keeps hteir verification key public
- which is preferred?

State

- **Stateful Protocols:** verificaiton and secret keys are updated every login
- **Stateless Protocols:** verification and secret keys do not change
- which is preferred?

2 Direct Attacks

Direct Attack Security

- The most basic idea is that the verification key of the server is the same as the password, just check if $vk = sk$
- Problems? If the server is compromised, the password is leaked
- basic solution 1: store a hash of the password on the server and compare those stored values
- the flaw is that one instance of eavesdropping reveals the password!

Online Dictionary Attack

- if you think the password is from a common password, you can simply try the most common passwords
- a common defense is to double the response time every two failed login attempts, to get around this, adversaries might employee botnets

Offline Dictionary Attack

- If a server is compromised, the attacker can steal the password database
- You can simply run the hash function locally and compare to the database until you crack the dictionary passwords
- You don't even need to crack the server's security, you can by the old server hard drives from the company, often you can recover data from those
- 1 in 2 passwords are suspected to be contained in a password dictionary, using random passwords is extremely important

Public Salts

- salts are just random bits that are hashed along with the verification key and stored in the clear
- It makes cracking each individual password take as long as the entire input space if the salts are long enough
- using salts can prevent most pre-processing/offline dictionary attacks, but doesn't defend against weak passwords

Secret Salts

- secret salting is when you assign a very small random value to the end of a password before hashing and storing it in addition to the public salt
- this means the server itself has to brute force the secret salt every login attempt, if it is kept sufficiently short, it can not directly impact the user experience while adding randomness to otherwise weakish passwords

Slow Hashfunctions

- an individual user logging in will not notice a hash function that is several orders of magnitude slower than Sha-256, but it will signfiicantly slowdown an attacker
- in addition to the actual slower speed of these hash funciton, if we make their memory access hard, then they will be less vulnerable to hardware attacks

3 Eavesdropping Attacks

Hashbased One Time Passwords (HOTP)

- these require both state and secret keys
- both a prover and the verifier maintain counters
- the prover sends the verifier a hash of the passwor with its current counter and the counter, and then increments the counter
- the verifier then checks the counter is greater than its counter, checks the verification against its hash of the password and the counter, and then increments its counter
- weaknesses: passwords are valid until the next counter increment, we need to keep the user's count and the server's count synchronized

Timebased One Time Passwords (TOTP)

- the counter is incremented based on time, perhaps every thirty seconds
- when a user tries to authenticate, the server calculates the counter based on the current time
- a lot of phone authentication apps work by setting a one time password and then using timing to generate the current one time password
- the weakness of this system

4 Active Attacks

S/key

- leverages the concept of hash chaining
- a secret key is sampled and a verification key is generated by creating a hash chain of $H^n(sk)$
- verification key is given in the clear
- to authenticate, prover sends the $H^{n-1}(sk)$ to the verifier who then hashes it one more time to verify, sets the new verification key to the most resent proof by prover
- security rests on the irreversibility of the hash function