Types Of Authentication

Authentication is the act of proving our identity to another participant.

Conceptually, authentication follows a 3 step challenge-and-response process. In the first step, the client claims an identity. Usually this is done by supplying a username, as we are all familiar with. After the username is supplied the remote server will respond with a challenge, often times a request for an agreed-upon password. The user then supplies the proof, which in most cases is the secret password.

Authentication Falls into 3 Categories in Order to Prove One's Identity:

- Something you have (I.e. your phone or key) … the drawback of this is this item can be stolen and used by someone else to authenticate on your behalf
- Something you are (i.e. your face, voice, fingerprint)… the drawback of this is how confidence can one trust this type of authentication schema
- Something you know (i.e. password) …. drawbacks include be brute forced, difficult to recall which can lead to password reusability

PARTY/ TICKET PERSON ANALOGY:
Suppose you purchase a ticket to an event (which the ticket is registered in your name)
Then you take that ticket to a bouncer (along with your ID which is something you have to authenticate yourself)
The name on ticket matches to name on ID thus person X must look like this (something you are)
Then I'll do a visual check and let you in
And I'll provide u with a wristband (session cookie) so u can come back in whenever and I don't have to authenticate u again
But it will expire after party is done so it's a limited authentication mechanism.

WHAT YOU KNOW:

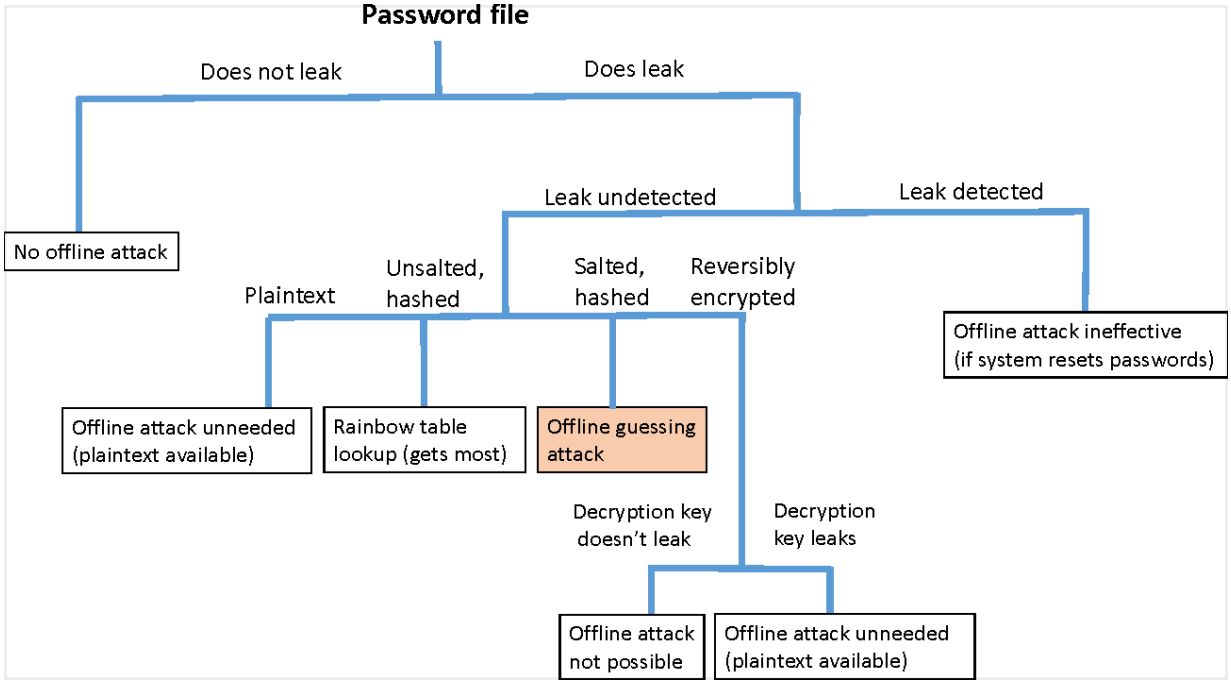
Rainbow Tables is a way to find password values by doing a hash and reduce function instead of pre-computation. The reduce function is not the inverse of a hash but rather it's just mapping of the hash to some other random plaintext (no relation whatsoever). Conceptually, one starts with a string then maps it to it's respective hash ... if that hash is not related to any target, then you reduce or map to some other string then hash... you keep doing this process over until you get some hash string that equates to what's in DB (thus that previous reduce string is your password)

Hash Functions are use to store passwords given some of its properties: output is irreversible, slight changes to the input lead to a brand new output. Hashing passwords are susceptible to dictionary attacks which a malicious user precomputes hashes of common passwords and then when a password store is leaked, they can check for commonalities and what the actual plaintext password is. Salting prevents this by adding random bits to the input each time then hashing it thereby making the precomputed hashes of a dictionary attach rendered useless as the same input + random salt always leads to a brand new hash so malicious hacker cannot work backwards.

TLDR - Salts render pre-computation useless thus it can counter dictionary attacks and rainbow tables

NOTE: Even with these mechanisms still in place, password leaks still occur due to mass adoption problems amongst companies (password policies are not enforced for example) and poor user password hygiene is to blame

OFFLINE ATTACK CHART!

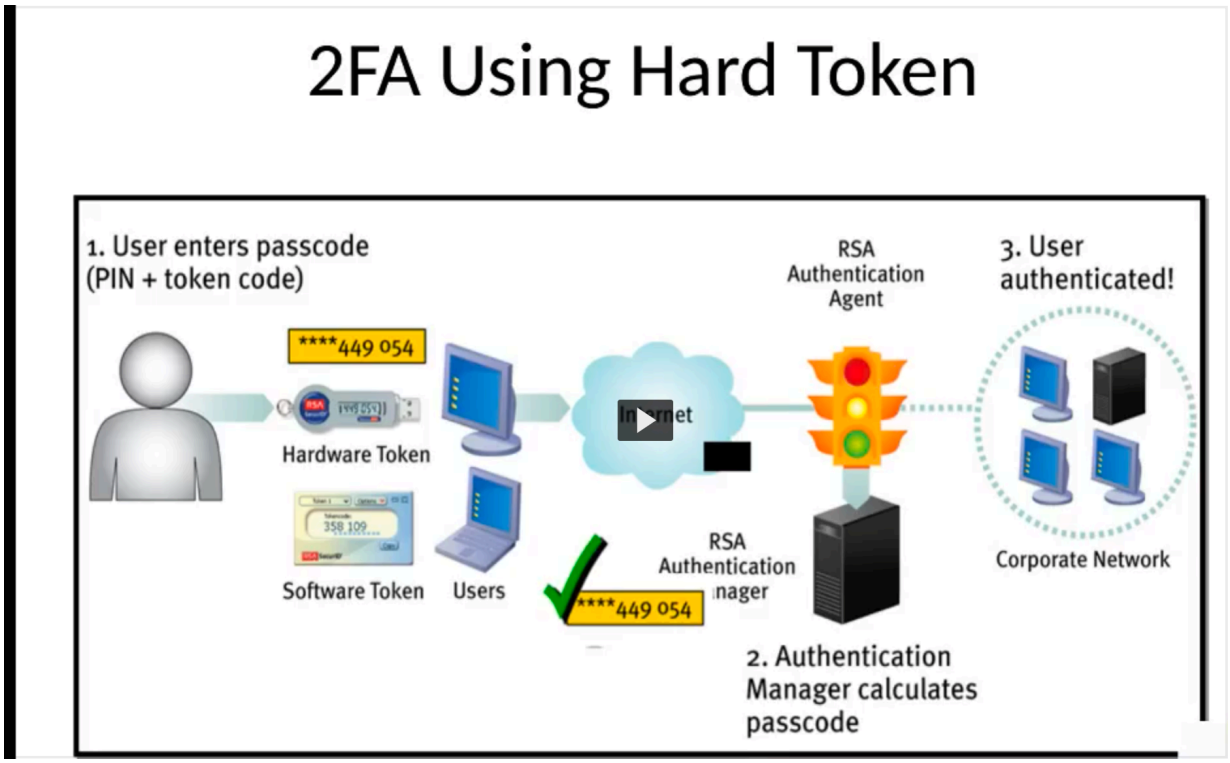


Password Policies:

- Used to enforce certain rules upon password creation upon account signup (there is no one standard password policy FYI but median strength for banks are 31 bits, 20 bits for .com sites, and 43 for .edu and .gov sites)
- The issue with this is that the burden gets placed on the user but password managers alleviate that as they just remember one password and that grants them access to all other passwords

MultiFactor Authentication:
A combination of 'something you know' + 'something you have' that serves as extra layer to authenticate yourself. So if something you know is compromised, then the malicious actor cannot enter a domain pretending to be you without access to a physical device.

Tokens (hard, mobile, soft) are physical means to provide secondary layer of auth. Hardware OTP (one time pass) are typically time based physical mechanisms while software ones are event based OTP.



OTP GENERATION:
The hardware token and AUTH server move at the same cadence (i.e. has the same seed or digit value, same time interval to update this seed, and same algorithm to generate new seed values)

HOW ARE PASSWORDS CRACKED:

Facts:
Passwords containing 6 lowercase passwords = 26^6 = ~308 million combinations
Passwords containing 8 characters (letters, numbers, symbols, etc.) = (26 alpha lower + 26 alpha upper + 10 digits + 40 symbols) ^ 8 = ??

ENTERPRISE LEVEL AUTHENTICATION:

Kerberos - a protocol used to authenticate people on a network based on tickets.

Initial user authentication

The process begins when a registered user requests an initial ticket from the authentication server (AS). The AS then sends to a ticket granting service (TGS) the ticketing information, and returns a ticket generating ticket (or TGT) to the user. The user then contacts the TGS and supplies this TGT, as well as the entitiy that the user wants to contact (like a file server, for example). The TGS will respond with a ticket that the file server while recognize as valid. In order for this to work, the TGS needs to be aware of all of the resources available on the network or in the domain.

Attacks that Kerberos Prevents

Kerberos protects against quite a few attacks. As an authentication method, Kerberos prevents imposter attacks, since it uses the user's secret key as proof of identity. In addition, Kerberos prevents eavesdropping and man in the middle attackers. This is because sensitive information is never sent on the network unencrypted, and the only information that touches the network unencrypted is information that's potentially publicly known. In addition, only valid users can generate needed output (such as answers to challenge messages), based on their secrets. Finally, Kerberos protects against replay attacks due to things like timestamps and lifetime fields that are present in tickets.

Kerberos Problems

Like everything else, Kerberos has its flaws. Unfortunately there is a single point of failure in the KDC. If the KDC is unavailable, you cannot log into services on the network. Additionally, there can be problems with time synchronization. Since tickets only last a certain amount of time, if the time synchronization is off users may not be allowed into a resource they should be able to get into. This can be minimized with NTP, but cannot be solved completely.