6️⃣

Chapter 6 - Attacking Authentication

Attacking Authentication

- Should be simple in theory however is often done wrong and can be abused in many ways.

Authenticaation Technologies

- HTML forms-based authentication
- Multifactor mechanism
- Client SSL certs
- HTTP basic and digest auth
- Windows-integrated auth using NTLM or Kerberos
- Auth services
- 90% of authentication is done via HTML forms

Design Flaws in Authentication Mechanisms

Bad passwords

- common to encounter applications that allow passwords to be:
  - very short **or** blank
  - Common dictionary words or names
  - The same as username

    - Still set to a default value

Brute-Forcible login

- You can be rate limited
- rate limiting can be bypassed if it is poorly implemented, eg a cookie counting number of times you have attempted login can be deleted
- even if you are rate limited, it may still give you a different message if you got it correctly

Verbose failure messages

- If the error message is specific to which piece of info was wrong, you can enumerate
- even with generic error messages, usernames can sometimes be enumerated with how long the request took, eg if it was found it may take longer

Vulnerable Transmission of Credentials

- HTTP eavs droppers may reside in:
    - on the user’s local network
    - within the user’s IT deparment
    - within the user’s ISP
    - on the internet backbone
    - Within the ISP hosting application
    - within the IT deperatment managing the application
- Even with HTTPS credentials may not be safe:
    - as a query string may be logged in browsing history, web server logs, proxies/vpns
    - even if they use a POST form it may be passed to a 302 redirect URL without user knoledge
    - bad apps may store creds in the cookie

Password change functionality

- periodic enforced password changes can mitigate vulnerabilities
- verbose error messages indicating if the user is valid
- often forget to rate limit existing password field
- often runs into error due to complex action

Forgotten password functionality

- opens up the possiblity for account takeover
- security questions are not secure
- if the app has a “hint” functionality, users will often just write their password or something very obvious
- one time passwords or challenges can be vulnerable
- sometimes the application will drop the user into an authed session after successful change.

“Remember me” functionality

- check if there is a persistant cookie specific to a user, this can be abused
- the persistant cookie, may not use a username but an ID, check if this ID is predictable.
- even if it is not gusable, the cookie may be vulnerable to xss

User Impersonation Functionalitiy

- Some apps may allow an admin to control a user’s account, some vulns that come from this:
    - may be a hidden function which is not protected, but known only by some
    - may trust user submitted info via cookies
    - any weak logic may allow vertical priv esc
    - there may be a backdoor password or pyramid key that works with any user (very bad)

Incomplete Validation of Credentials

- some bad apps truncate passwords and only store/use x amount of first chars

- some bad apps are case incensitive
- some bad apps strip unusual chars

Nonunique Usernames

- if the same two users have the same password it may break
- more efficient brute forcing

Predictable Usernames

- If an app makes the user’s name, eg customer1, customer2, etc, new usernames can be easly guessed

Predictable initial Passwords

- sometimes apps distribute initial passwords to a user, there can be problems here if they can be guessed

Insecure Distribution of credentials

- creds or activation urls that are sent via sms or email or often not time restricted or not changed by the user (if not forced)

Implementation Flaws in Authentication

Fail-Open Login Mechanisms

- This is when the backend will authenticate you if there is an error, this can be caused by your user input
- This session may not be tied to a user and thus you may have less functionality

Defects in Multistage Login Mechanisms

- some multistage logins make unsafe assumptions at each stage with reference to earlier stages:
  - the app may assume that if a user is at stage 3, then stage 1 and 2 must have been passed
  - the app may only validate a piece information on one stage, if it can be changed on later stages it may not be sanitized.

    - the app may not deal with sessions between stages indicating that the user is the same, check for hidden fields or weak cookies etc

Insecure storage of credentials

- clear text passwords in db
- unsalted hashed passwords in db

Securing Authentication

Use strong credentials

- minimum password requirements should be set
- usernames should be unqiue
- any system generated creds should contain entropy
- passwords should allow for long strings and special chars

Handle credentials secretively

- all data transmitted to and from server should be protected
- Credentials should never be put in url get parameters
- Credentials should never be transmitted back ot the client
- hash all creds

Validate credentials properly

- use catch-all exception handlers that delete any session
- auth code should be heavily reviewed
- any user impersonation functionality should be heavily reviewed
- when it comes to multistage auth:
    - all data about progress from stages should be kept server-side and never transmitted to client
    - the same data should never be able to submitted twice

    - the first task on every stage should be to check if the prior stages are correct
    - if a wrong answer was submitted, carry on with the other questions and give a generic error message at the end.

Prevent Information Leakage

- use generic error messages as much as possible to stop enumeration
- enumeration can happen in other ways than specific error messages, eg typograpgic details, different HTTP code responses, information in hidden HTML forms
- when rate limiting someone do not allow them to know how many requests it takes to be rate limited, nor the length of time they will be rate limitted for. This can lead to automation.

Prevent Brute-Force Attacks

- dont just think about logins, changing the password, recovery situations, etc can be brute-forced
- some high security apps need out of band steps to reactive an account after ratelimit
- if an account is temp suspended, their credentials should not even be check, an error should be returned immedietly without processing
- password spraying is often not checked (one password, wordlist of usernames)
- Captchas are good at preventing any kind of automation

Prevent Misuse of the Password Change Function

- function should only be accessible from within an authed session
- username's should never be submitted, there is no need for one user to change another's password
- re enter current password to stop xss, click jacking, etc
- user should be notified out of band, ie email

Prevent Misuse of the account recovery function

- high security apps like banks may use out of band recovery, eg speaking with someone on the phone
- never implement hints
- The best automated solution is a unique, time restircted, single use, unguessable url

Log, Monitor, and Notify

- log all authentication-related events, eg login, logout, password change, password reset, account suspension, account recovery, failed and successfull login attempts.
- logs should contain username, ip and relevent details. Never any passwords
- Anomalies should be processed in real time
- users should be notified out of band for any critical security event. eg password change
- users should be notified out of band for any frequently occouring attempts. Eg if one person is being targetted alot, they can change their password.