MOD-3

3.1 Design Flaws in authentication (13)

Implementation flaws in authentication (3)

3.2 Securing Authentication

3.3 Hydra

~~Thon~~ John the Ripper.

SDS | Page No. | Date

Authentication

- Authentication lies in the heart of the application, to protect that against malicious attack. Frontline defence against unauth. access.
- A wide range of technologies are available, for authentication mechanism like,
  - Html form based,
  - multifactor i.e. passwords + physical tokens,
  - SSL certificates or smartcards, etc

Design flaws in authentication Mechanisms

1) **Bad Passwords** ⇒

- Many web app never take into consideration, the quality of user passwords and may allow vague passwords like very short, blank, dict. words, or may leave the default one.

2) **Brute Force login** ⇒

- As attacker sees login, guessing username & password is the first ~~thes~~ thing they may do.
- Thus if multiple attempts are allowed to login, brute forcing the pattern of passwords until a correct one may be done.

→ Any attacker uses automated techniques to attempt to guess password. But now in some cases, client side controls are employed to stop password-guessing. but problem comes when an session is storing the failed attempts, during new session, the count of failed attempts would be zero.

→ For that, app locks out the acc. after set amt of guesses.

3) Verbose failure Message

- A typical login form requires the user to enter 2 pieces of information generally username & password, some apps, require more.
- When an login attempt fails → we can infer that one piece of info was wrong, but if app mentions what part was wrong, then this can be exploited by attacker.

"password is incorrect" ← wrong practices.

And if username is known, bruteforce can be done to get in the system (finding password).

4) Vulnerable transmission of cred's:

- If an app uses unencrypted HTTP connectn to transmit login credentials, a ~~be~~ eavesdropper, who is positioned on the network can intercept them.

→ eavesdropping can be done at users local network, users Isp, etc.

- even if login occurs of HTTPS, then also creds may still be disclosed if app. handles them in unusual manner:

⇒ If credentials are transmitted as query string parameters, as opposed to in the body of POST req., they can be logged into various places like users browsing history. If attacker tracks this, ~~can~~ may get the creds.

⇒ Redirect method is used instead of more conventional method of POST. Typically when POST is used, data is checked with the server. But in some cases, application redirects the user to diff URL & passing the creds as query string parameters. in URL itself.

⇒ webapps sometimes store user creds in cookies, for "remember me", thus making the creds vulnerable, thus attacker can get the user cooking thus compromising the data.

5) Password change functionality

- Many webapp do not provide this functionality but is necessary. for two reasons
  → changing of passwords lower ~~less~~ downs the threat of password compromise & reduces window if password is compromised.
  →
  → If user knows his password is compromised, he can change the password.

6) Forgot password functionality

- Forgot pass. functionality involves presenting the user with a secondary challenge. & is much often easier for an attacke to attack then to guess password.
  → Sometimes they ask questions like mention mothers name, but this generally has a very less amt of ans & either are available somewhere.

→ sometimes on main login page brute force attack may not be possible but on forgot pass. page, there wont be any restriction thus bruteforcing can be done here

→ set by user itself

⇒ sometimes hints are provided for password recover, but these hints are too easy to be guessed by attacker.

⇒ Reasonable secure mean of doing password forgot is by sending reset links to email. & to be done in set interval of time..

7) "Remember me"

- Implemented as a convinience to user, where they do not need to reenter the credentials.

⇒ some times "remember me" is done using cookies only, that can easily be broken by attacker.

⇒ sometimes an ~~session~~ user identifier is kept in cookied instead of username, but if any one session (user identifier is leaked, that can easily predict other values too.

8) using impersonation functionality

- (verbally authenticate an user)
- Some apps allow a facility for previliged user to impersonate other user & access their data.
- Some impersonating functionality is implemented as a 'backdoor' password, that can be submitted to standard login page with any username to authenticate as user.

Attackers are likely to discover such passw. & if if backdoor pass is matched before actual passw. Bruteforce might get 2 hits.

9) Incomplete validation of creds

- Some apps require or they put constraints on setting the password by user like length of password, etc.
  & correspondingly, poorly designed authentication mechanism do not care about this.

⇒ some apps may truncate pass, & verify only first n characters of a password. or may perform case insensitive check, etc.

10) Non Unique Usernames

- Some application allow user to get their usernames & don't care if the username is being repeated. This becomes a design flaw because
  → one user who shares username with another user may also happen to select same password in later pass. changes. That may create a conflict.
  → attacker exploits this behaviour to carry out bruteforce.

11, 12) Predictable usernames & predictable Initial Passwords

- Automatically generated usernames, can lure the attacker and also might get exhausted as ~~app~~ they are app. generated.

- In some apps user are assigned passwords, meaning the method of generation of password may get leaked/attacked by attacker.
  → common when a copy of password is sent to user.

13) Insecure dist. of creds

- Many app. generate creds for newly registered accs, and passes the creds to users. via mail, SMS etc.
- Sometimes if the mail / SMS does not reach the exact person as marked, may compromise security & even if it reaches the correct person, it is less likely that they are going to change the default password.

Implementation flaws in authentication

1) • Even a well designed auth system may likely be vulnerable due to wrong implementation. & may lead to info. leak, by passing logins
• The implementation faults are more hard to detect

a) fail-open login Mechanism

- If a call to db.getUser (uname, passwd), throws an exception,

↑
one who is going to check if user is there in dB or not

lets say exception of null value, ~~this~~ n but login still gets done, this means data was not provided yet login was successfull.

2) Defects in Multi-state login mechanisms

- some apps use various stages in login,
  a) username password, then
  b) memorable PIN / word.
  c) physical token / captcha

- It requires user to fill out various details but at first it wants user to identify themselves & subsequent stages perform various authentication checks.

⇒ But sometime it may assume that once user reaches stage 3, it has correctly filled 1 & 2 & allows logging in.

⇒ Appi may assume that data on all 3 stages is of same user and may not verify that, & can be used by attacker.

⇒ secret question method → mothers name (discussed earlier).

3) Insecure storage of creds

- If an app stores login creds insecurely, even though there aint any inherent flow in authentication process itself.

- common to find webapps where user creds are stored insecurely in the DB. (passwords as clean txt, or hashed using std. algo). leading to various vulnerabilities, allowing attacker to access data - via SQL injection.