Authentication

Kameswari Chebrolu

Department of CSE, IIT Bombay

Outline

- Authentication Background
- Causes of Vulnerabilities and their Impact
- Common Vulnerabilities and safeguards
  - Vulnerabilities in password-based login
  - Vulnerabilities in multi-factor authentication
  - Vulnerabilities in other authentication mechanisms
- Overall Best Practices

Background

- Clients make requests and web server responds
  - GET request method is read-only but POST, DELETE, PUT can make changes in server
  - Need to ensure actions are restricted to those with right permissions
    - E.g. A student user should should not be able to delete or even view assignments posted by other students
    - A e-commerce customer cannot access someone else's cart or use their pay balance for own purchase
  - Above is handled by **Access Control**

Access Control System

- Terminology
  - Subject: entity that requires access to a resource (e.g. users, batch-job, daemons)
  - Object: resource to which access is controlled (e.g. file, command, terminal)

- Access Control System manages
  - **Subject authentication (our focus now)**
  - Session management (covered under background)
    - Identify which subsequent requests were made by the same subject
    - Session hijacking attacks covered under client based XSS and CSRF attacks)
  - Authorization (upcoming)
    - Check if the subject has permissions to access the object
    - Revoke access when required
  - Audit (not covered)
    - Record the access for future review

Authentication vs Authorization

Authentication: verify user is really who they claim to be

Authorization: verify whether a user is allowed to perform a given action

Authentication

- Is HTTP request being issued by a specifically-identified user?
- Based on the following factors
  - Something user knows (e.g. password)
  - Something user has (e.g. mobile phone)
  - Something user is (e.g. fingerprint)

- Typically implemented as a shared secret between user and web server in the form of a password (a single factor)
- But 2FA (two factor authentication) is gaining popularity (e.g. bank settings)
  - One has to enter password, then OTP (sent to or generated in mobile phone)

- Details of Implementation
  - Login forms implemented in HTML via forms
  - Supplied username and password sent to server as a POST request (often over HTTPs)
  - Server calculates hash of password and checks with value stored in database
  - Server given user a session token
  - Browser sends token to server on every request
  - Server will use the token
    - To authenticate each request
    - To tie this requests with past requests to maintain state

Outline

- ~~Background~~
- Causes of Vulnerabilities and their Impact
- Common Vulnerabilities and safeguards
  - Vulnerabilities in password-based login
  - Vulnerabilities in multi-factor authentication
  - Vulnerabilities in other authentication mechanisms
  - Vulnerabilities in Oauth authentication
- Overall Best Practices

Causes of Vulnerabilities

- Weak authentication mechanisms that do not protect against brute-force attacks
- Logic flaws in implementation that allows authentication to be bypassed

Impact

- Can be very severe, especially compromise of high-privileged accounts (e.g. system administrator)
  - Not just full control of app, but can also access internal infrastructure
- Compromise of low-privileged account can also increase attack surface

Outline

- ~~Background~~
- ~~Causes of Vulnerabilities and their Impact~~
- Common Vulnerabilities and safeguards
  - Vulnerabilities in password-based login
  - Vulnerabilities in multi-factor authentication
  - Vulnerabilities in other authentication mechanisms
- Overall Best Practices

Brute Force Attacks

- Attacker uses trial and error to guess user credentials
  - Need to guess both **username** and **password**
  - Often automated using educated guesses/popular wordlists (dictionary attacks)

Username

- Often easy to guess
  - E.g. firstname.lastname@example.com or admin or administrator or root
- Username enumeration: attacker sends username and observes changes in server's behavior (via response)
  - Can leverage login page or registration forms

- Server response:
  - Status codes: Difference in HTTP status code (200, 401, 404 etc)
    - Defense: Return same status code regardless of guess
  - Error messages: Returned error message are different
    - “Both username AND password” incorrect or
    - “Only password” incorrect
    - Defense: Use generic “identical” messages in both cases
      - “Incorrect username or password entered”
      - Note sometimes developers may make mistakes in error messages, one character difference can reveal info!
        - “Incorrect username or password entered”
        - “Incorrect Username or Password entered”

- Response times: Difference in response time between wrong and correct guess
  - E.g. Calculate hash of password for checking only if username is valid
  - Can increase response time if guess of username is correct
  - Attacker can enter a very long password to increase this delay further
  - Defense: calculate hash of password even for invalid usernames

Passwords

- Two types
  - Online attacks
  - Offline attacks

Online Password Attacks

- Try a large number of username/password combinations against the login page
  - Often limited by the speed of the network (3 – 5 login attempts per second)
  - Password brute forcing is a function of strength of the password!
    - Will involve many failed guesses before success
    - Need to be more sophisticated than simply iterating through all possible combinations

Safeguards

- Enforce use of high-entropy passwords; Change passwords regularly
  - A minimum number of characters
  - A mixture of lower and uppercase letters
  - At least one special character
  - Inspite of above, users use (weak) password they can remember and try to fit into the password policy

- Make it difficult to automate the process and slow down the rate at which an attacker can attempt logins
  - Lock the account after too many failed login attempts
  - Block the IP address after too many login attempts (same account) or in quick succession across accounts (rate limiting)

Circumventing Account Lock

- Locking an account offers some protection against brute-forcing a specific account
- Does not protect against accessing any radom account
- Make a list of (likely valid) candidate usernames
  - Obtained via username enumeration or some common/popular list

- Decide on a very small list of passwords that have high probability of hit (across usernames)
  - Ensure number of passwords is under the max login attempts permitted before locking
  - Limit of 3 attempts → max 3 password guesses
  - Password spraying: try a few common passwords against many accounts
- Try the list of selected passwords (through some automated process) against username in the candidate list
  - Success with one single username → account compromised

- Doesn’t help against credential stuffing attack as well!
  - Credential stuffing: Use previously stolen usernames and passwords to gain unauthorized access to accounts.
  - Use a big dictionary of username, password pairs
    - Stolen in data breaches
    - Many people reuse the same username and password across websites
  - Each username is only attempted once → limit never reached
  - One attack can compromise many different accounts!

Circumventing IP Block

- Circumvention depends on how implemented?
- Suppose the counter to IP block is reset if login successful. How to leverage this?
  - Add valid login:passwd (of attacker's) after every 2 guesses of attempted login → limit is never reached
  - Suppose attacker is trying to crack user1's password
    - user1 passwd1; user1 passwd2; attacker passwd; user1 passwd3 etc

- Another implementation:
  - Too many login requests in a short window → block IP address
  - Unblock after some time or manually by an admin or by user after some successful captcha
  - How are requests measured? Rate of HTTP requests sent per IP address
- Can bypass if you can guess multiple passwords with a single request
  - E.g. if implementation supports json objects

```
{
 "username": "Chotu",
 "password": ["123456",
"password!", "IruleWorld", ...]
}
```

Offline Password Attacks

- Offline password attacks: Recover one or more passwords from a password storage file or database
  - Storage file: Windows: SAM; Linux: /etc/shadow
  - Database: dump in the form of [username, H(password)] or [username, salt, H(password|salt)]
  - How obtained?
    - Via attacks such as SQL injection, SSRF, Directory traversal, Information disclosure etc

- More advantageous than online attack
  - Invisible to security team and altering mechanisms (lockouts don’t happen)
  - Limited by speed of computer (not network)
    - Can attempt a few million/billion password guesses per second

- Servers save [username, H(password)]
  - Why hash and not plain text? Database breaches can reveal all passwords!
    - Not very strong against offline password cracking attacks
    - E.g. SHA256, 1 billion guesses per sec, cracking half of 100 million accounts via dictionary attacks can take minutes to hours
- Often salted, server stores [username, salt, H(password|salt)]
  - Makes it more difficult (days/months) to crack passwords offline

- Cracking Passwords: Need to generate candidate passwords
  - Candidates can come from various sources
    - commonly used passwords (from data breaches)
    - dictionary words
    - permutations of known passwords
    - randomly generated strings etc
  - For each candidate,
    - Hash them using same algorithm/process used by target
    - Compare it with hash obtained from the target
    - If match found, candidate password is likely correct!

- Time to crack a password depends on many factors
  - Complexity of the password
  - Hashing algorithm used
    - Precomputed hash tables (rainbow tables) are often used to speed up computation
  - Any extra security implemented via salting or key stretching
    - Key stretching (e.g. bcrypt) does hashing multiple times making it more time-consuming and resource-intensive to crack passwords
  - Hardware resources available to the attacker

- Mitigation:
  - Strong Password Policies: Encourage users to choose strong, complex passwords
    - These will not show up in candidate lists!
  - Use Salting/Key Stretching: slows down has calculation and make it more difficult for attackers to crack!

Outline

- ~~Background~~
- ~~Causes of Vulnerabilities and their Impact~~
- Common Vulnerabilities and safeguards
  - ~~Vulnerabilities in password-based login~~
  - Vulnerabilities in multi-factor authentication
  - Vulnerabilities in other authentication mechanisms
- Overall Best Practices

Multi factor Authentication

- Multi factor authentication provides better security
- Two Factor Authentication (2FA) is most common
  - Involves a traditional password and a temporary verification code
- Note: verifying same factor in two different ways does not provide much security
  - E.g “What user knows” being verified twice
    - Password and OTP sent to email which is accessed via login credentials

- Verification codes are handled via “what user has” i.e. some physical device
  - In high security settings, a dedicated device is often provided
    - Device stores secret key and connects to the computer and helps you access websites online
  - Dedicated mobile app that generates a verification code
    - E.g. Google Authenticator
  - Send verification codes to user's mobile phone via SMS
    - Not as secure since code can be intercepted
      - SIM swapping attack: Attacker can obtain a SIM card with victim’s phone number

Bypassing 2FA (Via Implementation Flaws)

- Implementation flaw-1: Improper verification of verification code
  - User shown first page for entering username and password
    - Internally, state of user changed to logged in
  - User shown second page for entering verification code
  - But, server does not check verification completion and can load other internal pages
  - Attacker: Don’t enter verification code, just navigate to other pages
    - These other pages normally accessible only after valid login

- Implementation flaw-2: Mallory is the attacker who has a valid account, but is trying to login as Alice
  - User shown first page for entering username password

```
POST /login/step1 HTTP/1.1
Host: vulnerable-website.com
….
username=mallory&password=qwerty
```

```
HTTP/1.1 200 OK

(some session cookie set)
```

  - User shown second page for entering verification code
    - Normally below would happen
      - GET request causes the server to generate a verification code
      - In Post Mallory will submit the verification code she received

```
GET /login/step2 HTTP/1.1
Cookie: session-id=xyz; verify=mallory
```

```
POST /login/step2 HTTP/1.1
Host: vulnerable-website.com
Cookie: session-id=xyz; verify=mallory
…
verification-code=1234
```

- But server doesn't verify "**same**" user is completing the second step, so attacker can do following
  - Below requests sent in second page
    - Note first step is same as before i.e. log in was done by Mallory
  - Brute-force the verification code
    - Why? Server will generate a verification code for Alice and sends it to her
    - Mallory has to guess this code, so try all combinations via POST
  - Note password of Alice was not even needed!

```
GET /login/step2 HTTP/1.1
Cookie: session-id=xyz; verify=alice
```

```
POST /login/step2 HTTP/1.1
Host: vulnerable-website.com
Cookie: session-id=xyz; verify=alice
…
verification-code=0123
```

- Safeguard: log out user after a certain number of incorrect verification codes
- Circumvention: Can automate the process via macros (part of ZAP, burp-suite tools)
  - You will get logged out, but you will automatically log back in and try again and again
  - Easier if the verification code does not change across logouts
    - Server pinned OTP, attacker trying various combos
  - But can try same verification code each time and check → will take many attempts
    - Attacker pinned OTP, server generating different code each time

Outline

- ~~Background~~
- ~~Causes of Vulnerabilities and their Impact~~
- Common Vulnerabilities and safeguards
  - ~~Vulnerabilities in password-based login~~
  - ~~Vulnerabilities in multi-factor authentication~~
  - Vulnerabilities in other authentication mechanisms
- Overall Best Practices

Other Authentication Mechanisms

- Most websites provide additional functionality to manage accounts
  - Remember me/keep me logged in
  - Change/reset password
- Websites normally look at vulnerabilities in the login page and overlook other functions
  - Particularly relevant where users can create own accounts and can try these additional pages for vulnerabilities

Keeping Users logged in

- Even after closing a browser session, users often like to stay logged in
- Facilitated via a simple checkbox
  - "Remember me" or "Keep me logged in"
- How implemented? A persistent token is stored at user end

- Possession of token can bypass entire login process → tokens impossible to guess
- Vulnerability: predictable tokens
  - E.g. Concatenation of username and timestamp
  - Sometimes token are generated by hashing predictable values
    - If attacker can create their own account, they can examine cookie carefully
    - Can recover via dictionary attacks
- Techniques such as XSS can also be used to steal such tokens

Resetting User Password

- Users can forget password → need to provide a way to reset them
  - Need to make sure the actual user is resetting, not someone else
- Some send new password by email
  - Not very secure since inboxes are not designed for secure storage!
- Send a unique password reset URL to users
  - E.g. http://vulnerable-website.com/reset-password?user=victim-user
    - Attacker can change the user parameter to refer to any other username where they can potentially set their own password

- Generate hard-to-guess tokens
  - E.g. http://vulnerable-website.com/reset-password?token=a0ba0d1cb3b63d13822572fcff1a241895d893f659164d4cc550b421ebdd48a8
  - Implementation flaw: Maintain a pool of currently valid tokens (no user state maintained)
    - Attacker tries to reset own password and gets a token
    - Attacker then constructs a POST request where
      - Attacker shares own token
      - But changes user to someone else along with new password
    - Server checks token validity from the pool and uses username from POST and changes the password of that user!

- When user visits URL, system should check whether this token valid against this user!
- Further, token should expire after a short period of time or immediately after the password has been reset

Changing User Password

- Reset password: User doesn't remember password
- Change password: user remembers password
- Changing involves entering current password and then the new password twice
  - Like a normal login page, website checks that username and current password matches
  - Then it updates the password
- Similar brute force attacks as on login page are possible here as well

Brute forcing password via change password URL:

- Implementation is as under:
  - Wrong current password; two entries for new password match → account locked
  - Wrong current password; two entries for new password do not match → current password incorrect
  - Correct current password; two entries for new password do not match → new passwords do not match

- Attack:
  - Choose a username whose password you want to crack
  - Set two non matching new passwords
  - Enumerate various passwords under the current password field
  - Whatever password results in “new passwords do not match” is the correct password

Outline

- ~~Background~~
- ~~Causes of Vulnerabilities and their Impact~~
- Common Vulnerabilities and safeguards
  - ~~Vulnerabilities in password-based login~~
  - ~~Vulnerabilities in multi-factor authentication~~
  - ~~Vulnerabilities in other authentication mechanisms~~
- Overall Best Practices

Best Practices

Not possible to enumerate all safeguards. Some general principles:

- Always send login data over encrypted connections (https)
- Ensure no username or email addresses are disclosed through publicly accessible profiles (leads to username enumeration)
- Implement an effective password policy
  - Encourage users to set strong passwords via password strength checkers!
  - High security settings, make users change passwords periodically and do not allow use of older passwords!

- Use identical, generic error messages, and make sure they really are identical!
- Return same HTTP status code with each login request
- Make the response times in different scenarios as indistinguishable as possible
- Implement robust brute-force protection
  - Implement strict, IP-based user rate limiting
  - Include CAPTCHA test with login attempt after a certain limit is reached!

- Triple-check verification logic
  - Don't forget supplementary functionality (reset/change password)
- Use multi-factor authentication if feasible!
  - Much more secure than password-based login alone
  - But don’t use the same factor across all!
  - Use a dedicated device or app that generates the verification code directly (rather than SMS)
  - Logic also should be without flaws!

Real Life Examples

- Improper Authentication - any user can login as other user with otp/logout & otp/login: https://hackerone.com/reports/921780
- Bypassing GitLab Two-Factor Authentication: https://hackerone.com/reports/128085/
- Auth Bypass = Username Enumeration + Login Brute Force: https://hackerone.com/reports/209008

Summary

- Authentication an important part of access control
  - HTTP protocol supports basic and digest
  - But often implemented via POST payload followed by token based session management
- Discussed various vulnerabilities in password, multi-factor based authentication and in other mechanisms
- Covered overall best practices as well

References

- https://portswigger.net/web-security/authentication
- https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html