Authorization

Kameswari Chebrolu

Recap: 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
  - Session management
    - Identify which subsequent requests made by the same subject
  - **Authorisation (our focus now)**
    - **Check if the subject has permissions to access the object**
    - Revoke access when required
  - Audit
    - Record that access for future review

Authorization

- Helps meet CIA criteria of the resource
  - Information not exposed to unauthorized parties
  - Information not modified by unauthorized parties
  - Information always available to authorized parties

- Can have very serious impact if not handled well!
  - Facebook 2018 major data breach

Model

- Good to choose and follow a model that is a right fit for the application
  - Without a model that defines rules, difficult to check implementation is correct!
- Various models!
  - Discretionary Access Control (DAC)
  - Mandatory Access Control (MAC)
  - Role Based Access Control (RBAC)
  - Attribute Based Access Control (ABAC)

Discretionary Access Control (DAC)

- Owner of data decides who can access the data (objects)
- Used by nearly all operating systems
  - E.g. A file may have following access (ls -l filename): rwxr-xr–
  - Owner can read, write, or execute it, group can read and execute, others can only read
  - Owner can change the permissions on files on their own → discretionary policy.
- Information specified as an access control matrix [Lampson 1971]

User	File 1	File 2	File 3
Kanti	R, W, X	R, _, _	_, _, _
Bhushan	R, _, _	R, W, X	R, _, X
Dhanush	_, _, _	R, W, _	R, _, X
Jyoti	R, _, X	_, _, _	R, W, _

- Information from the matrix can be stored either by columns or by rows
- If stored by column, it is called access control list (ACL)
  - Windows and Unix use such an implementation
  - Most SQL databases also implement ACL-based authorization
    - Account used to connect to database determines which tables can be read/updated/changed!

- If stored by row, it is called a privilege list or capability list
  - Android uses a privilege list
  - App can be granted the right to access the network, GPS, phone, etc
- Highly granular (access rights defined at individual resource/function/user) but very complex to design and manage

Mandatory Access Control (MAC)

- Assign access rights based on regulations by a central authority
- Philosophy: information belongs to an organization (rather than individual members) → organization should control the policy
- Most common implementation is Multilevel Security (MLS)
  - Used in military, government, and intelligence applications
- Difficult to implement in regular applications

Role Based Access Control (RBAC)

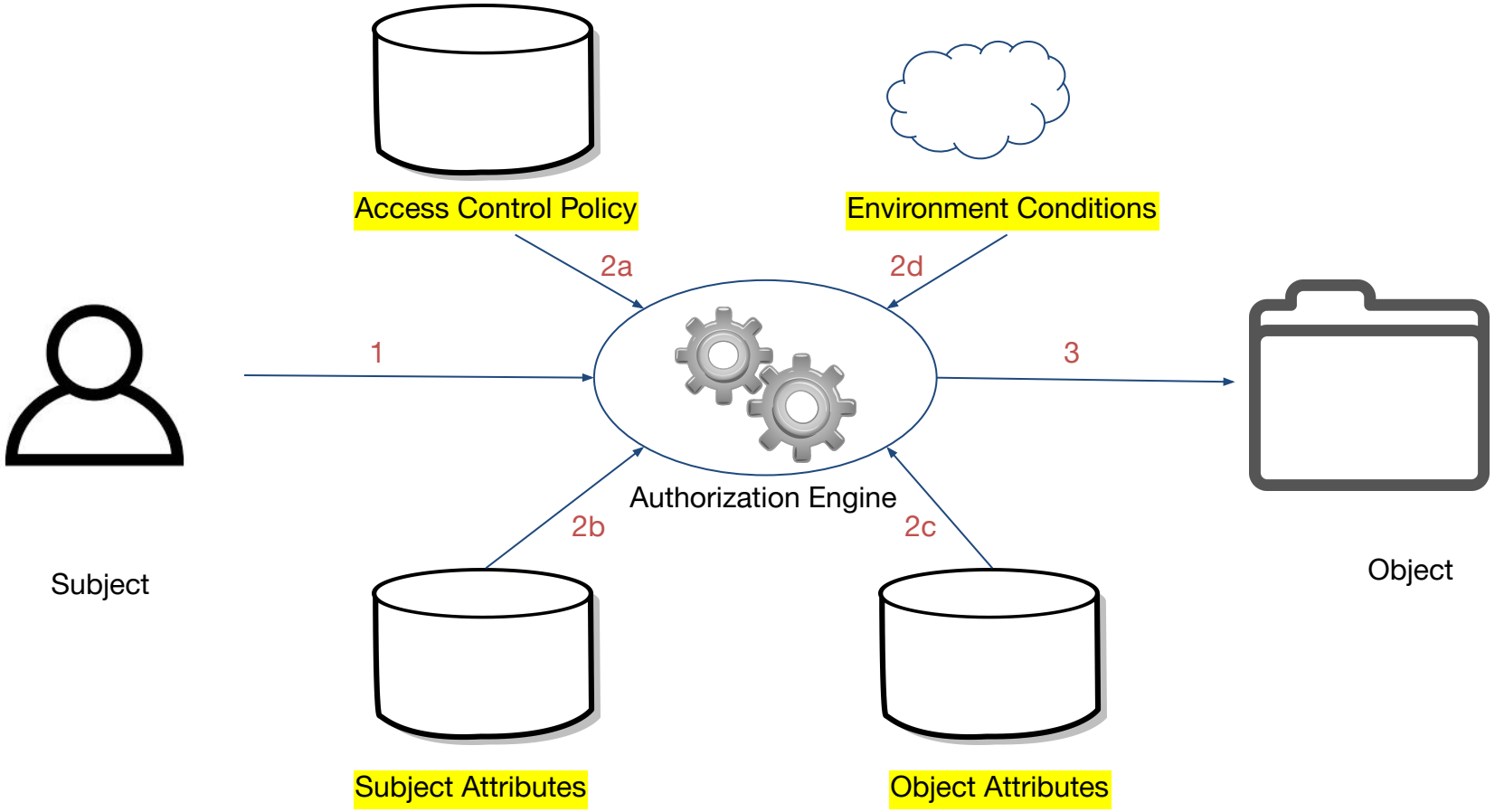
- Security policies in real-world are dynamic
  - Access rights change over time as responsibilities of users change
- Role: a job function or title
  - E.g. normal user or administrator
- RBAC changes the underlying subject–object model
  - Access is implicitly predetermined by the person assigning the roles to an individual
  - Access is explicitly predetermined by the object owner based on the privilege associated with each role

- Earlier: Access control policy is a relation on subjects, objects, and rights
- Now: Access control policy is a relation on roles, objects, and rights
- Each subject may be assigned to many roles
- Each role may be assigned to many subjects
- Roles are hierarchical
  - E.g. Instructor has all the rights of TA and some more. TAs have all the rights of students and some more

- Extensively used in web applications, cloud
  - E.g. Amazon's Identity and Access Management (IAM)
- Can be complex
  - Policies can contradict each other, creating conflicts
  - Users can belong to many groups with overlapping concerns
    - Difficult to debug why a system is making certain access control decisions!

Attribute Based Access Control (ABAC)

- Access Control based on "assigned" attributes of subject, object, environment conditions; and a set of policies
  - Attributes specified by a name-value pair
  - Environment conditions:
    - Operational or situational context in which access requests occur; Independent of subject/object
    - E.g. current time, location of a user, current threat level etc
  - Policy expresses a complex Boolean rule that can evaluate many different attributes



ABAC: Access Control Mechanism

Access Control in Web Applications

- Web application often employ both identity based access control (ACL lists) and also RBAC
  - RBAC provide access to different functionality (students, TAs, Instructors, administrators) → **vertical access control**
  - IBAC (DAC) provides protection of discrete records (grade records of individual students) → **horizontal access control**
    - One student cannot access other student records (role is same: student)

- Context-dependent access controls also used in some cases
  - E.g. cannot submit assignment after deadline
    - Restrict access to functionality and resources based upon the state of the application
  - Can be managed via ABAC, but often implemented using RBAC/IBAC via some extra state
  - ABAC should however be preferred in general!
    - Gives lot more flexibility, is robust and easier to manage

Broken Authorization (Vulnerabilities)

- Vertical Privilege Escalation
- Horizontal Privilege Escalation
- Vulnerabilities in other places
  - Multi-step, referrer header, location etc

Vertical Privilege Escalation

- Application does not enforce any protection of sensitive functionality
  - E.g. https://vulnerable-website.com/admin gives you admin access if you can guess the URL
    - URL often will not show up under a normal user's browsing → developer thinks this is fine
    - But can be revealed under robots.txt or URL path can be brute-forced by guess+wordlist (admin, root, administrator etc)

- Implementation flaw-1: Make URL unpredictable
  - Security by obscurity; not a good practice in general (will leak somehow)
  - E.g. https://vulnerable-website.com/admin-99004
    - Developer can leak URL via javascript code
      - E.g. JavaScript constructs the user interface for end user based on the user's role

```
<script>
 // ...
 if (isAdmin) {
   // ...
   // Create a new anchor element
   var adminPanel = document.createElement("a");
   // Set the href attribute to the URL of the admin panel
   adminPanel.setAttribute("href", "https://insecure-website.com/admin-99004");
   // Set the text displayed on the anchor element
   adminPanel.innerText = "Admin panel";
   // ...
 }
</script>
```

- Implementation flaw-2: URL is public but a user-submitted parameter is used to do access control
  - Determine user access rights at login, store it in a cookie
    - E.g. role=2 (implies its normal user)
    - Cookie sent with each request
  - Not a good practice, attacker can control the parameter
    - Can change the parameter (role=1; admin role)

- Implementation flaw-3: Access control implemented in frontend via configuration in javascript
  - Configuration specifies that for HTTP method POST and the endpoint /admin/User, access is denied for users with the role "students"

```
const accessControlConfig = {
    "DENY": {
        "POST": {
            "/admin/User": ["students"]
        }
    }
};
```

- Attacker: Non-standard HTTP headers can be used to override URLs in original request
  - E.g. Use of X-Original-URL or X-Rewrite-URL
- Frontend looks at POST and / is allowed but backend application will overwrite / with /admin/User and execute it

```
POST /  HTTP/1.1
X-Original-URL: /admin/registerUser
```

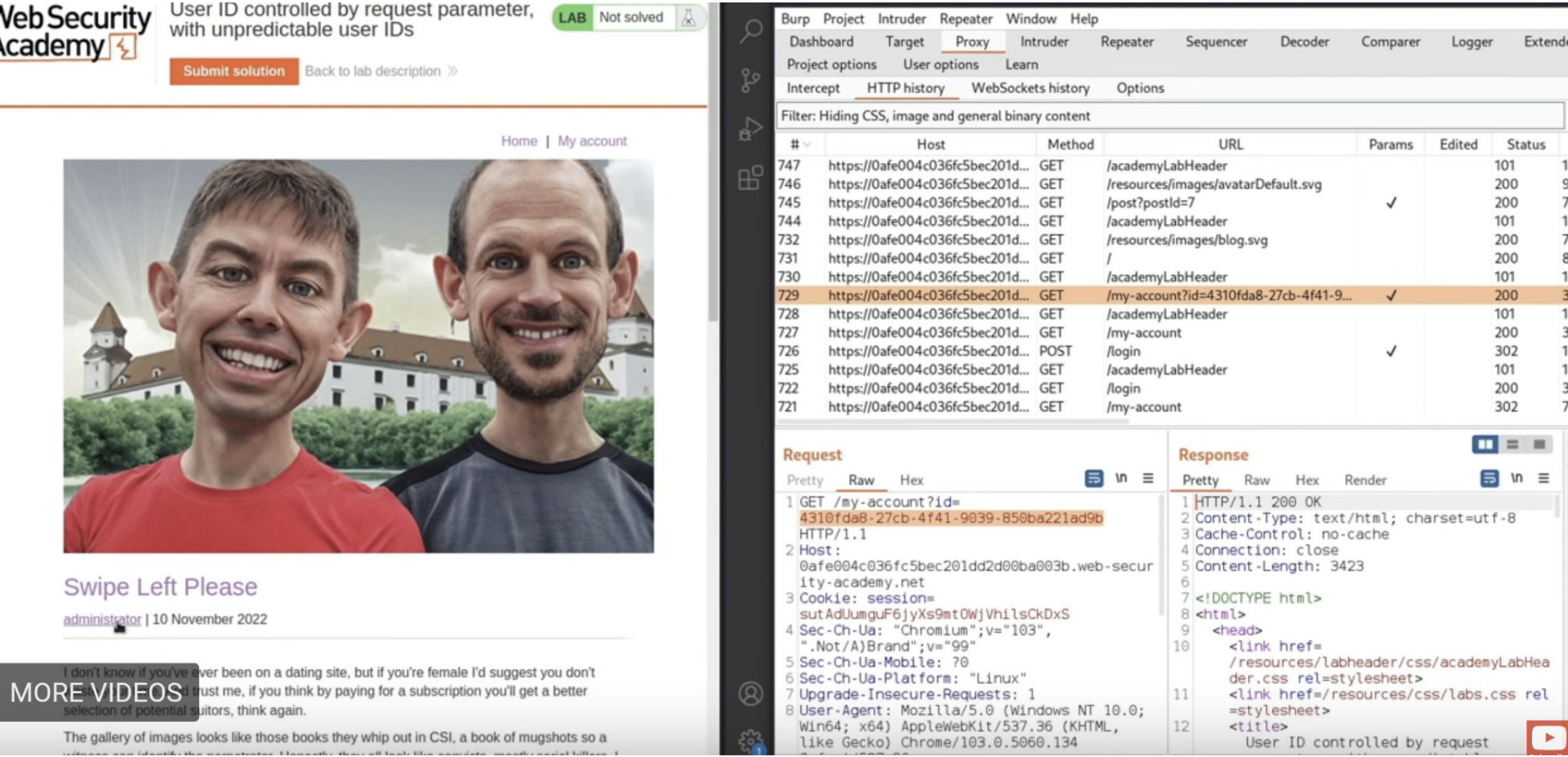
- Similarly one can also convert a POST into a GET
  - Assuming provided application supports functionality via various methods
  - Since platform/frontend checks only POST, it will allow GETs

- Other flaws:
  - Inconsistent capitalization
    - a request to /ADMIN/USER may map to the /admin/User endpoint at backend
    - But access control may treat these as two different endpoints and fail to enforce the first one
  - Similarly,
    - /admin/deleteUser.php vs /admin/deleteUser
    - /admin/deleteUser vs /admin/deleteUser/
      - Can bypass access controls by appending a trailing slash to the path

Horizontal Privilege Escalation

- A user is able to gain access to resources belonging to another user (same hierarchy level)
- E.g. A user may access their account details via say https://vulnerable-website.com/myaccount?user=ravi
- An attacker can modify the user parameter value to that of another user
  - https://vulnerable-website.com/myaccount?user=rashmi

- Similar flaws as earlier
- E.g. Use unpredictable IDs i.e. globally unique identifiers (GUIDs) to identify users
  - Like before, security by obscurity bad idea!
  - GUIDs can be disclosed elsewhere (e.g. blog posts, discussion forum messages, reviews etc)



From port swigger

Insecure Direct Object Reference (IDOR)

- Type of vulnerabilities seen so far are called IDOR
- Occur when an attacker can access or modify a reference to an object that should not be accessible
- Happen when **user-controller parameter** values are used to access resources or functions directly
  - E.g. https://vulnerable-website.com/myaccount?user=ravi

- Can be used for both horizontal and vertical privilege escalation
  - https://vulnerable-website.com/myaccount?user=rashmi
  - https://vulnerable-website.com/myaccount?user=admin
- Another example:
  - https://vulnerable-website.com/assignments/1234.zip
  - Can download assignments of other students by guessing numbers

Miscellaneous

- Multi-step Access control:
  - Important functions are implemented over a series of steps
  - E.g. 1) Click on delete user, 2) Confirm you really want to delete user, 3) Then delete
  - Developer implements rigorous access controls over step 1 but not step2
    - Assumes can reach step2 only after step1
  - Attacker can gain unauthorized access by skipping first step and directly submitting request for second step with required parameters

- Referer header based access control
  - Referer header added to requests by browsers
  - Indicates which page (URL) initiated the request
  - E.g. www.vulnerable-website.com/admin (main page)
  - www.vulnerable-website.com/admin/next (nextpage)
    - This page endpoint only inspects the referrer header and allows request if referer header correct
    - Assumes proper access control over the main admin page
  - Attacker can forge referrer header
- Similar issues with access controls based on user's geographical location (e.g IP address)
  - Easy to spoof or use proxies (relay agents)

Testing

- Vulnerability scanners like Burp Suite or ZAP can be used to check for IDORs
  - Set a payload on the id parameter, and choose a numerical payload to increment or decrement
  - Set a payload on the user parameter, and choose a list of popular user names
  - Change in status code 403 (access denied) vs 200 (ok) can tell if guess was successful

Proper Implementation

- A centralized access control where decisions are made is best!
  - Easier to validate, all in one place
  - Use principle of least privilege
  - Enumerate all types of users, resources and operations (such as read, write, update, etc)
  - Use ABAC based model if possible else role/identity
  - After the app has been deployed, periodically review policy

- When making access control decisions, important to vet identity data
  - Should not rely on anything in the HTTP request besides session cookie!
    - A malicious user can tamper with anything else in the request
- Important to also use unit tests that make assertions
  - Who can access certain resources and, importantly, who shouldn't access them
- Good to also use an external team to perform penetration testing
- Add audit trails to help with debugging and also later forensic analysis
  - Log files or database entries that record whenever a user performs an action
  - E.g. User example@gmail.com logged in at 01:12:06 2022-01-18

- Ensure access is also done over static resources and ones that aren't normally discoverable
  - Developers often do no access control over hidden pages!
  - But hacking tools can quickly enumerate private URLs; some can be guessed as well!
- Make sure access control rules account for any timing requirements
  - Some websites upload reports with URLs of the form /reports/<month-year>)
  - Cheaters have been known to check such URLs ahead of the rest of the market
    - Financial watchdogs have charged such companies large fines!

Summary of Best Practices

- Do not do "security by obfuscation"
- Deny access by default
- Permission should be validated correctly on every request
- Use centralized access control if possible (ABAC is a good model in complex web applications)
- Don't name target pages with meaning
  - Use key-value pairs, with unpredictable values, that don't leak
- Thoroughly audit and test access controls

Real Life Examples

- Horizontal Privilege Escalation https://hackerone.com/reports/98247/
- Steal API tokens (horizontal escalation): https://hackerone.com/reports/95552/
- Vertical Privilege Escalation: https://hackerone.com/reports/159387

Summary

- Authorization a very important part of access control, helps meet CIA requirements
- Many models: DAC, MAC, RBAC, ABAC
  - Web apps typically uses DAC (IBAC) +RBAC
- Attackers attempt horizontal and vertical privilege escalation (IDOR attacks)
- Covered various implementation best practices

References

- https://portswigger.net/web-security/access-control
- https://cheatsheetseries.owasp.org/cheatsheets/Authorization_Cheat_Sheet.html
- https://www.researchgate.net/publication/338993052_A_Survey_of_Access_Control_and_Data_Encryption_for_Database_Security