Attacking Access Controls

- Access control vulnerabilities are conceptually simple: The application lets you do something you shouldn’t be able to

- **Common Vulns**

→ Access-controls can be divided into 3 categories

⇒ Vertical

• Allow different types of users to access different parts of app's functionality. In simplest case it is user and admin. In more complex cases it may involve combination of specific user roles.

⇒ Horizontal

• Horizontal controls allows users to access a certain subset of a wider range of resources of the same type.

⇒ Context-Dependent

• Ensures that user's access is restricted based on current state of the application

→ The main attacks against access controls

⇒ Vertical privilege escalation

• Occurs when a user can perform fucions that his assigned role does not permit him to.
• Ex: Normal user performing admin functions

⇒ Horizontal privilege escalation

• Occurs when a user can view or modify resources to which the user is not entitled.
• Ex: Using a mail client to read other user's email

⇒ Buiness Logic Exploitation

• User can exploit a flaw in the app's state to gain access to a key resource.
• Ex: A user may be able to bypass payment in a shopping checkout sequence

→ It is common for horizontal attacks to lead to a vertical attack. Although these attacks allow an authenticated user to escalate privs.
→ In its worst cases, unauthenticated users can access resources only available for privileged authenticated users.

→ Conpletely Unprotected Functionality

⇒ Often secure functions are not displayed anywhere on a low priv accounts's interface.

⇒ This doesn't stop the attacker from gaining access, it is merely security through obscurity.

⇒ If a restricted panel's URL is guessable, then gaining access to this resource is easy.

⇒ Even if the resoource is not referenced anywhere, client side code such as JS can be a quick way for attackers to learn about hidden resources in the app.

→ Direct Access to Methods

⇒ Direct acces to API methods and params that are used in invocation of these methods are to be restricted

⇒ Ex: http://ww38.wahh-app.com/public/securityCheck/getCurrentUserRoles

• Seeing a Link like this, probe for similar named methods such as getCurrentUserPermission, getAllUserRoles,getAllUsers etc.
• Notice the naming conventions and follow them, here it a Java Camel Case.

→ Identifier Based Functions

⇒ When a function of an app is used to gain accessto resources, it often uses an identifier.

⇒ Usually, these identifiers are either guid (randomly generated) or predictable numbers

⇒ Either ways these resources might be vulnerable.

⇒ In the guid case, since these are served in the URL, they can be retrieved from places where URL's get stored, browser hostory, caching server, proxy servers etc.

⇒ If the identifiers is just a sequential number, then it is seriously vulnerable if access controls are not implemented.

⇒ Ex: http://ww38.wahh-app.com/ViewDocument.php?docid=1280149120

• In this link, the docid can be changed to see if any document not owned by you is visible to you.

⇒ Similar ids can be assigned for user roles and senitive functions. Probe these ids to check if you can execute sensitive functions.

→ Multistage Functions

⇒ Many functions take a multi-stage approach in implementation

⇒ Often access controls are checked in the earlier stages of these functions and the later stages “assume” that the user does have permissions to access them.

⇒ If an attacker is able to circumvent this, the attacker can possibly perform a vertical or horizontal attack to escalate their privileges

→ Static Files

⇒ Most cases involve users to request dynamic pages to access resources

⇒ But often some static reources are present on web servers which are served to the user.

⇒ If an attacker get hold of a predictable static resource like this, the attacker can try to access other static resources that may be present on the server.

⇒ Ex : https://wahh-books.com/download/9780636628104.pdf

- In this link the ISBN of the book is the static filename, as an attacker if you gather other book's ISBN and launch an automatic attack,the attacker will be able to access books they have not paid for.
- Static log files and other sensetive data are particularly prone to this attack

→ Platform Misconfiguration

⇒ Some apps use controls at the web server or app platform layer to control access.

⇒ This is avert legitimate way to control acces but the platform level configuration take the form of three rules

- HTTP request method
- URL path
- User role

⇒ Often denied request can be circumvented htrough change of HTTP request method.

- Ex: GET to POST, POST to GET and HEAD to replace GET.

⇒ Since most application APIs are agnostic to req mthod. Simple by changing the method and request to a method, we can access sensitive functions

⇒ In some cases unrecognized HTTP headers are passed down to a GET req handler. Hence by changing the request to an arbitrary value, we might be able to circumvent the access control

→ Insecure Access Control Methods

⇒ Some apps make access control decisions based on req params or other conditions that are within an attacker's control

⇒ Parameter-Based Access Control

- The user's role or access level is determined on login and retransmitted every time in a hidden form field or cookie or a string param
- Ex: https://wahh-app.com/login/home.jsp?admin=true
- In this URL, the admin param decides the admin privs of a user. If a normal user tries to transmit this in their req they might be granted admin privs over the app.
- These types may be difficult to detect without using the app as a high priv user. But using the hidden content diccovery methods it is posssible to find and leverrage some

⇒ Referer-Based Access Control

- Other unsafe apps use the Referer Header as the the basis for making access control decisions.
- Expecting admin functions to be refered from the admin page.
- But this is fallacious as the Referer header is in control of the attacker and can be easily changed to escalate privs

⇒ Location-Based Access Control

- Many buisnesses have a regulatory or buisness requirement to restrict access to resources depending on user's geographic location.
- In the most common of which is the IP addressused to determine the geolocation of the user.
- To Bypass use
  - ◇ Web Proxy
  - ◇ VPN
  - ◇ Mobile device with data roaming
  - ◇ Manipulation of client side mechanisms for geolocation

- **Attacking Access Controls**

→ Before Attacking any actual access controls, review the results of application mapping. Understand what the app's actual requirement and focus on those areas.

→ Testing with different user accounts

⇒ Most effective way is to use different user accounts and check controls

⇒ If a higher priv account is available, first browse through using the high priv account and try to access these resources using a low priv account

⇒ You can use Burp suite to map out contents of an application using two different user contexts.

⇒ Compare sitemaps and see if you can exploit something

→ Testing Multistage Processes

⇒ Many processes use a multi stage pattern to execute some functionality.

⇒ Test each stage of these function for access control vulns.

⇒ Reach points where the app might assume you are already authenticated to access resources and use a low priv user to try and access these.

⇒ Every Step and every request has to be tested

⇒ To test in different context you can use request in browser feature which doesnt change the cookie or session token headers.

→ Testing with limited access

⇒ Having only one account, try adding params and use the Referer header effectively to detecc any changes in behaviour.

⇒ Try checking for horizontal priv esc

⇒ Try guessing, generating or enumerating for ids that are used.

⇒ If they are guid, try guessing something of the same length and in a similar range of characters.

→ Testing Direct Access to Methods

⇒ Try to access unprotected API methods.

→ Testing Controls Over Static Resources

⇒ Test static resource URL(s). Check if you can access files that you do not own.

⇒ Check if static resources are accessible by unauthenticated users.

→ Testing Restrictions on HTTP Methds

⇒ Test other HTTP methods where csrf tokens are absent

- POST
- GET
- HEAD
- An arbirtrary header

- Securing Access Controls

⇒ Avoid Obvious Pitfalls

- Do not rely on user's ignorance of application URLs and identifiers used in the app
- The access control by itself should be enough to prevent unauthorized access
- Do not tryust any user sbmitted params for access control
- Do not assume users will access resources in a sequence
- Do not assume user transmitted data is not tampered

⇒ Best practices

- Evaluate and document the access control for every unit of app fucntionality
- Drive all access control decisons from user's session
- Use a central application component to check access control
- Undestand that ids are prone to tampering and hence only server side data is to trusted. IDs are to be reevaluated.
- Reauthentication and dual authorization for sensitive actions
- Log every event. Useful in detection and investigation

⇒ A Multilayered Privilege Model

- Add multiple layers to specificate access control for each resuource
- Programmtic control - The matirix of privileges can be used to progmatically apply a privilege modell. These can be fine grained and can emplay complex logic.

- Discretionary Access Control (DAC)

  ◇ Admins can delegate their privs to specific resources employing discretionary access control. This is a closed DAC control (Access is denied unless denied)

  ◇ Administrators also can lock or expire individual user accounts. This is an open DAC model, in which access is permitted unless explicitly withdrawn.

- Role-based access control (RBAC)

  ◇ Named roles containning different sets of privileges. Using roles to perform up-front access checks on user requests to quickly reject unauthorized access.

  ◇ If platform level restrictions are places then they should be default deny.

- Declarative Control

  ◇ Using external objects to restrict access to sensitive data

◇ Ex: Use different database accounts with varying privs ro access DB.
▪ Even if the attacker bypasses the access control implemented in the application, the db account that is in use will retstrict such action
▪ Another way is to implemnt deployment descriptor files, but they are rather blunt and cannot act on fine-grained privs in a large app.