WFP1: SQL injection

Example #1
- Try the following inputs to detect injection
  - ?name=root1234
  - ?name=root+++
  - ?name=root"
  - ?name=root’
- Which one causes the script to terminate without outputting the table?
- This is the character that breaks syntax
- Craft an input that dumps the entire user table

Example #2
- Filter now eliminates certain characters
- Find an alternative to the filtered character to dump all users. (URL encode it)

Example #3
- Now, all whitespace is filtered
- One can use C-style comments, however, to achieve the same effect and dump all users

Example #4
- The script uses PHP’s `mysql_real_escape_string` to filter characters to prevent injections using strings
- SQL statements can be injected to invoke underlying database functionality such as integer arithmetic evaluation
- For example the following causes SQL to search for an id of 2
  - id=1%2b1
- Use this method to dump the user with `id=5` without using the number 5

Example #5
- Injections into SQL statements can be done without strings
- For integer SQL parameters, one must perform further input validation
- This page uses the following broken filter
  ```php
  if (!preg_match('/^[0-9]+/', $_GET["id"])) {
    die("ERROR INTEGER REQUIRED");
  }
  ```
- Craft a statement that begins with a digit and dumps the entire user table. (Hint: Use the integer equivalent of ‘1’ = ‘1’)

Example #6
- Does this page’s filter improve anything?
  ```php
  if (!preg_match('/^[0-9]+$/m', $_GET["id"])) {
    die("ERROR INTEGER REQUIRED");
  }
  ```
- Dump the entire user table, then craft a regexp that will properly perform input validation to prevent this attack

Example #7
- Filter fixed to validate integers (both positive and negative)
  ```php
  if (!preg_match('/^-?[0-9]+$/m', $_GET["id"])) {
    die("ERROR INTEGER REQUIRED");
  }
  ```
However, the /m (PCRE_MULTILINE) option is enabled
- Only checks that one of the lines matches if multiple lines given
- Perform injection across multiple lines. Use the URL-encoding for \"n\" to dump all users

**Example #8**
- Page appears to order by name
  - Implemented in SQL as `ORDER BY name` or by `ORDER BY 'name'`
  - Break out of syntax and access SQL statements `ASC` or `DESC` to change the order in which results are returned

**Example #9**
- SQL statement now does not use backticks to delimit parameter
- Repeat #8, but inject directly without the backticks

---

**WFP2: SQL injections**

**Example #1**
- Script checks to see that rows are returned via an SQL statement that checks for a valid username and password
  - Test to see which characters break syntax in username
    - Inject an always true condition to login without proper credentials to get Success message

**Example #2**
- Script now ensures that exactly one row is returned before allowing login
  - Use the SQL LIMIT command to return exactly one row after the injection

**Example #3**
- Script now filters the single quote
  - It is still possible to break syntax by removing single quotes in the syntax via \\.
    - Consider a username of `foo\\`
    - SQL statement now does not use backticks to delimit parameter
    - SQL statement looks for username of `foo\'` and password= and now parses the password field as the rest of the SQL statement (that now includes the odd single quote at the end).
    - After escaping the quote in the username field, use SQL in the password field to obtain Success message

**Example #4**
- Examine the URL and identify SQL-ish commands
  - Use URL-encoding and knowledge of SQL to inject code that will dump all users from the database

**Example #5**
- Find what might be getting sent to SQL and modify the URL directly to dump all users
  - Then, craft a SQL UNION statement that does the same (important for #6)

**Example #6**
- LIMIT, ORDER BY, and GROUP BY are all SQL statements that can take their parameters from user input and are thus, injectable
○ Using the UNION statement, inject SQL that will dump all users in the table

- **Example #7**
  - The page queries for an `id` of 1, then returns all users that share the same name (user1)
  - The page can also cause a leak by allowing the output of arbitrary SQL statements to be included in error pages
  - Try the following parameters and see where their output winds up
    - `?id=extractvalue('%27<xml>%27,concat("/",{now})))`
    - `?id=extractvalue('%27<xml>%27,concat("/",(select version())))`
    - `?id=extractvalue('%27<xml>%27,concat("/",(select%20user())))`
  - What kind of other information on schema and user data can you obtain?

- **Example #8**
  - Second-order SQL injection
  - Insert a user with a name that, will create an injection when the user’s profile is clicked
  - Use a UNION statement as part of the user’s name
  - Insert a user whose profile will cough up the credentials of another user
  - Note: This is a shared database with all others in the class. Inject your own by adding your names in an SQL comment

- **Example #9**
  - PHP’s `mysql_real_escape_string` will escape problematic characters in strings with a backslash (0x5c)
    - Injecting a single-quote to see if the form works properly
  - Proper escaping can not occur if the database driver and the database use differing character sets
  - Specifically, if the front-end uses ASCII and the other uses a multi-byte character set such as GBK to support simplified Chinese, injecting a backslash can change the grouping of the bytes as they are interpreted leading to single-quotes getting through
    - **Example code**

```ruby
get '/' do
  ActiveRecord::Base.establish_connection SQLInjectionExample9.db
  res = []
  if params['username'] && params['password']
    begin
      sql = "SET CHARACTER SET 'GBK';"
    end
  else
    render template: 'index'
  end
end
```
  - Use a single-quote (0x27) as the username and submit the form
    - Examine the URL returned to see the URL-encoded request
    - No injection happens as, behind the scenes, the PHP script places the backslash escape character (0x5c) before sending it to the MySQL backend. The character sequence %5c%27 is parsed by the MySQL GBK backend as intended
  - Consider the UTF-8 sequence 0xE5 0x91 0xB5. URL-encode the following 3-bytes in the username in the URL to see the GBK character that is returned.
  - Now, in the username form field, cut and paste the GBK character into the field and then place a single-quote in front of the GBK character to see what is returned in the URL field
    - Try again, but place the single quote after the GBK character in the form
■ By injecting a backslash in one of these cases, the single quote is allowed through and syntax is broken
  ○ When %E5%91%B5%27 is used as a username, PHP escapes out the single quote with a backslash and sends %E5%91%B5%5C%27 to the MySQL GBK backend
    ■ %E5%91 is a valid GBK code as is %B5%5C
    ■ After GBK has consumed the backslash the PHP script has added, the single quote is allowed to break syntax
  ○ Use this to insert a true condition, and bypass authentication to obtain the Success message

WFP2: MongoDB injection
  ● Example #1
    ○ Use the canonical SQL injection technique, but with MongoDB syntax to generate an always true condition that allows you to login

WFP2: Mass Assignment
  ● Example #1
    ○ Reverse-engineer the user object that is used in the mass assignment
    ○ Set the administrator privileges directly upon account creation using a proxy or a Python script
  ● Example #2
    ○ Similar to #1, find another way to set administrator privileges to obtain administrator privileges
  ● Example #3
    ○ Login as user1 with password pentesterlab
    ○ Access the information of Company 2 by changing your own company_id to that of Company 2

DVWA
  ● SQL Injection (low and medium)

Homework
  ● Lessons: Injection
  ● Challenges: SQL injection #1-7, NoSQL Injection One, SQL Injection Escaping