Input Validation Vulnerabilities

Laurie Williams
williams@csc.ncsu.edu

Computer Science
NC STATE UNIVERSITY

Input Validation Vulnerabilities

Insecure Interaction Between Components

These weaknesses are related to insecure ways in which data is sent and received between separate components, modules, programs, processes, threads, or systems.

For each weakness, its ranking in the general list is provided in square brackets.

Rank	CWE ID	Name
[1]	CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
[2]	CWE-78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')
[4]	CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
[9]	CWE-434	Unrestricted Upload of File with Dangerous Type
[12]	CWE-352	Cross-Site Request Forgery (CSRF)
[22]	CWE-601	URL Redirection to Untrusted Site ('Open Redirect')

http://cwe.mitre.org/top25/#Categories

Computer Science
NC STATE UNIVERSITY

SQL Injection Vulnerability

1 CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')

Summary

Weakness Prevalence	High	Consequences	Data loss, Security bypass
Remediation Cost	Low	Ease of Detection	Easy
Attack Frequency	Often	Attacker Awareness	High

- Condition: When untrusted input is used to construct dynamic SQL queries.
- Consequence: Can be used to alter the intended query logic to access, modify, and delete data in the database, possibly including execution of system commands or creating a denial of service.

http://cwe.mitre.org/top25/#CWE-89

Computer Science NC STATE UNIVERSITY

SQL Injection - String Type Example

Login: John

Password: John1234

```
String query = "SELECT * FROM users
      WHERE login = '" + login +
      "' AND password = '" + password + "'";
```

Expected input:

```
SELECT *FROM users
  WHERE login = 'John'
  AND password = 'John1234'
```

Result: Returns John's user information

Computer Science NC STATE UNIVERSITY

SQL Injection – Tautology

Login: `' OR '1' = '1`

Password: `' OR '1' = '1`

```
String query = "SELECT * FROM users
      WHERE login = '" + login +
      "' AND password = '" + password + "'";
```

Expected input:

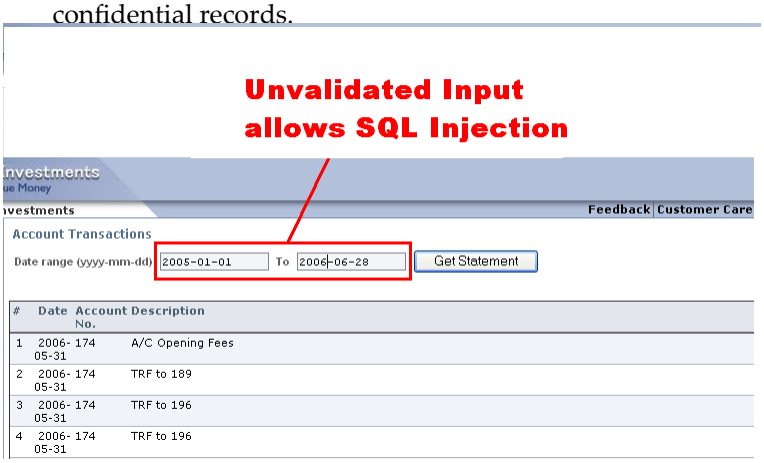
```
SELECT *FROM users
  WHERE login = '' OR '1'='1'
  AND password = '' OR '1'='1'
```

Result: Returns all user information in the users table

Computer Science NC STATE UNIVERSITY

SQL Injection – Date Type Example

Submitting SQL query logic instead of a valid date can expose confidential records.



From: www.itsa.ufl.edu/2006/presentations/malpani.ppt

Computer Science NC STATE UNIVERSITY

SQL Injection – Date Type Example

```
String query = "SELECT * FROM accounts
      WHERE username = '" + strUName + "'
      AND tran_date >= '" + strSDate + "'
      AND tran_date <= '" + strEDate + "'";
```

Expected input:

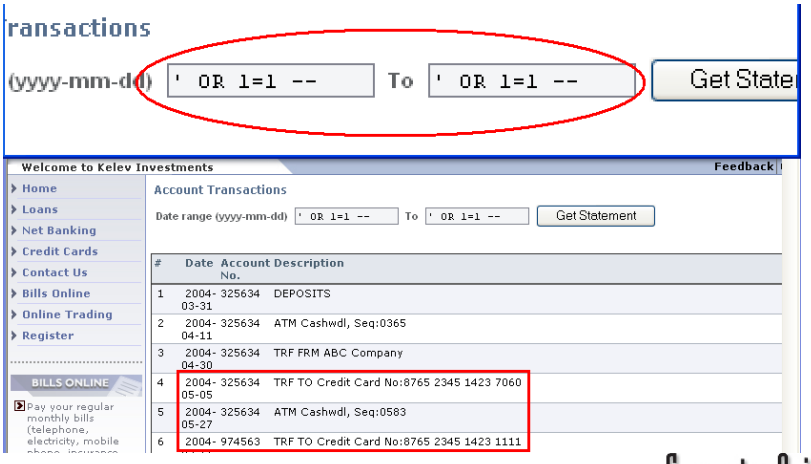
```
SELECT *FROM accounts
  WHERE username = 'John'
  AND tran_date >= '2005-01-01'
  AND tran_date <= '2006-06-28'
```

Result: Returns John's transactions between given dates

Computer Science NC STATE UNIVERSITY

SQL Injection – Date Type Example

Submitting SQL query logic instead of a valid date can expose confidential records.



Computer Science NC STATE UNIVERSITY

Modified from: www.itsa.ufl.edu/2006/presentations/malpani.ppt

SQL Injection – Date Type Example

```
String query = "SELECT * FROM accounts
      WHERE username = '" + strUName + "'
      AND tran_date >= '" + strSDate + "'
      AND tran_date <= '" + strEDate + "'";
```

Expected input:

```
SELECT *FROM accounts
  WHERE username = 'John'
  AND tran_date >= '' OR 1=1 --'
  AND tran_date <= '' OR 1=1 --'
```

Result: Returns all saved transactions

Computer Science
NC STATE UNIVERSITY

SQL Injection – Drop Table

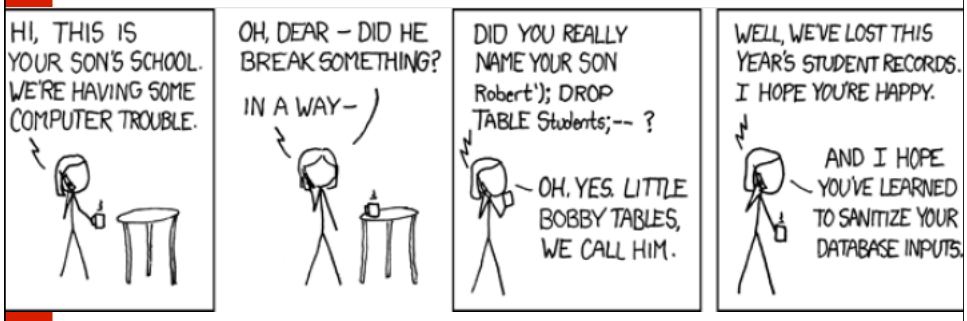
Could be any SQL command; add data; delete rows, etc.

- What if the attacker had instead entered:
  - **blah'; DROP TABLE prodinfo; --**
- Results in the following SQL:
  - SELECT prodinfo FROM prodtable WHERE prodname = **blah'; DROP TABLE prodinfo; --**
  - Note how comment (--) consumes the final quote
- Causes the entire database to be deleted
  - Depends on knowledge of table name
  - This is sometimes exposed to the user in debug code called during a database error
  - Use non-obvious table names, and never expose them to user
- Usually data destruction is not your worst fear, as there is low economic motivation

Jim Whitehead http://www.soe.ucsc.edu/classes/cmps183/Spring06/lectures/SQL%20Injection%20Attacks.pdf

Computer Science
NC STATE UNIVERSITY

Bobby Tables



Computer Science
NC STATE UNIVERSITY

Video

- http://www.youtube.com/watch?v=jMQ2wdOmMIA

Computer Science
NC STATE UNIVERSITY

Mitigation: Prepared Statement

- Pre-compiled parameterized SQL queries
- A setter method sets a value to a bind variable as well as performs strong type checking and will nullify the effect of invalid characters, such as single quotes in the middle of a string.
  - setString(index, input), sets the bind variable in the SQL structure indicated by the index to input

```
String custname = request.getParameter("customerName"); // This should REALLY be validated too
// perform input validation to detect attacks
String query = "SELECT account_balance FROM user_data WHERE user_name = ? ";

PreparedStatement pstmt = connection.prepareStatement( query );
pstmt.setString( 1, custname);
ResultSet results = pstmt.executeQuery( );
```

https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet

Computer Science NC STATE UNIVERSITY

Other Mitigations

- Example Database Frameworks
  - e.g. Hibernate framework … use createQuery()
- Use stored procedures
- Escaping non-alphanumeric characters
  - OWASP ESAPI Encoding module (database dependent)
- Input validation (whitelist)
- Will usually be caught by static analysis tools

- All mitigation methods have holes or can be misused.
- Defense in depth!

https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet

Computer Science NC STATE UNIVERSITY

Testing for SQL Injection

- Change the logic of a query

```
1' OR '1'='1 ' OR 1=1 -- ' OR 1=1 #  ') OR ('1'='1 --
```

- Denial of service using multiple statements

```
; DROP TABLE Users --
```

- SQL syntax depends on DBMS

Computer Science NC STATE UNIVERSITY

From http://www.owasp.org/index.php/Testing_for_SQL_Injection_(OWASP-DV-005)

OS Command Injection

2	CWE-78: Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')

Summary			
Weakness Prevalence	Medium	Consequences	Code execution
Remediation Cost	Medium	Ease of Detection	Easy
Attack Frequency	Often	Attacker Awareness	High

- Your software is often the bridge between an outsider on the network and the internals of your operating system. When you invoke another program on the operating system, but you allow untrusted inputs to be fed into the command string that you generate for executing that program, then you are inviting attackers to cross that bridge into a land of riches by executing their own commands instead of yours.

Computer Science NC STATE UNIVERSITY

http://cwe.mitre.org/top25/#CWE-78

Example 1

This example code intends to take the name of a user and list the contents of that user's home directory. It is subject to the first variant of OS command injection.

Example Language: **PHP** (Bad Code)

```
$userName = $_POST["user"];
$command = 'ls -l /home/' . $userName;
system($command);
```

The $userName variable is not checked for malicious input. An attacker could set the $userName variable to an arbitrary OS command such as:

(Attack)

```
;rm -rf /
```

Which would result in $command being:

(Result)

```
ls -l /home/;rm -rf /
```

Since the semi-colon is a command separator in Unix, the OS would first execute the ls command, then the rm command, deleting the entire file system.

http://cwe.mitre.org/data/definitions/78.html

Computer Science NC STATE UNIVERSITY

Cross-site Scripting (XSS)

4 CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')

Summary			
Weakness Prevalence	High	Consequences	Code execution, Security bypass
Remediation Cost	Low	Ease of Detection	Easy
Attack Frequency	Often	Attacker Awareness	High

- Condition: When untrusted input is used to construct HTML web pages.
- Consequence: An attack can control the appearance of a web site, transfer sensitive data, and hijack the session to take control of the user's account.

http://cwe.mitre.org/top25/#CWE-79

Computer Science NC STATE UNIVERSITY

Cross Site Scripting (XSS)

- Web application takes input from a user but fails to validate the input
- Input is echoed directly in a web page.
- Input could be malicious JavaScript, when echoed and interpreted in the destination browser any number of issues could result

Possible attack:

- When the victim is tricked to click on a crafted link (via web server or email), he is referred to the host in the URL
- The host processes the query string and echoes it to the victim's browser,
- The victim's browser executes the malicious script.

[1] "Build Security In" https://buildsecurityin.us-cert.gov/daisy/bsi/articles/knowledge/guidelines/342.html

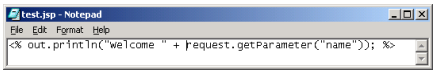
Computer Science
NC STATE UNIVERSITY

Cross Site Scripting

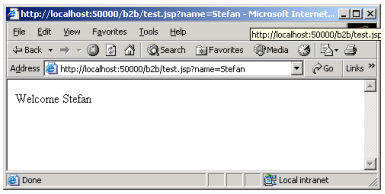
- Reflected XSS
  - Attacker-provided script is embedded in the web page generated by the server as an immediate response of an HTTP request.
- Stored XSS
  - Attacker-provided script is stored to a database and later retrieved and embedded in the web page generated by the server.

NC STATE UNIVERSITY

Cross Site Scripting – Reflected XSS

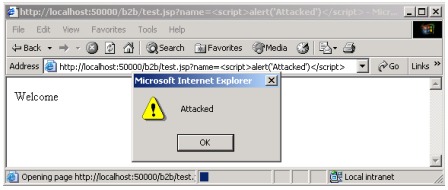


http://myserver.com/test.jsp?name=Stefan




```
<HTML>
<Body>
Welcome Stefan
</Body>
</HTML>
```

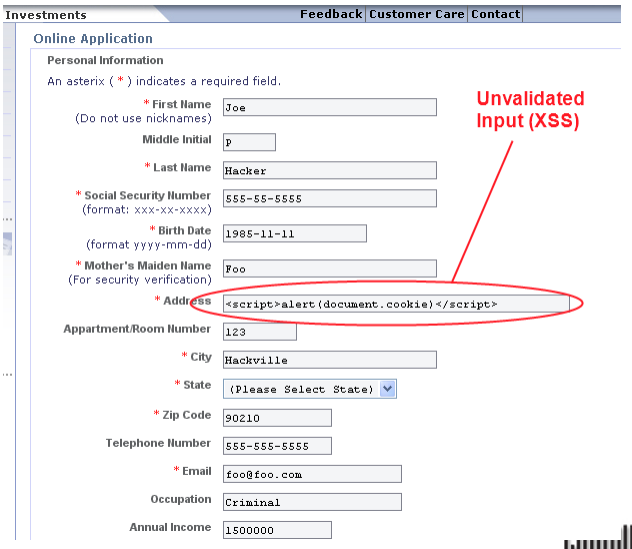
http://myserver.com/welcome.jsp?name=<script>alert("Attacked")</script>




```
<HTML>
<Body>
Welcome <script>alert("Attacked")</script>
</Body>
</HTML>
```

Computer Science
NC STATE UNIVERSITY

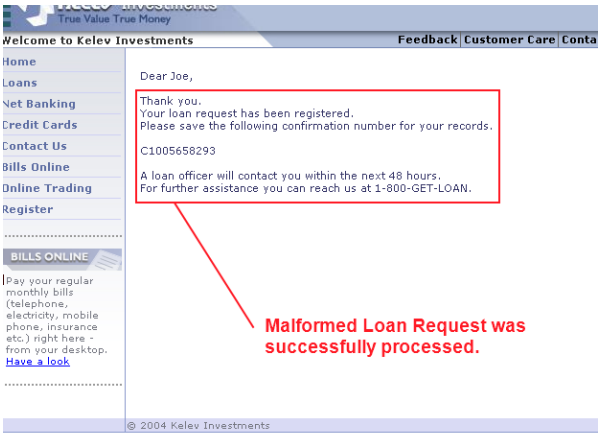
Cross Site Scripting – Stored XSS



Computer Science
NC STATE UNIVERSITY

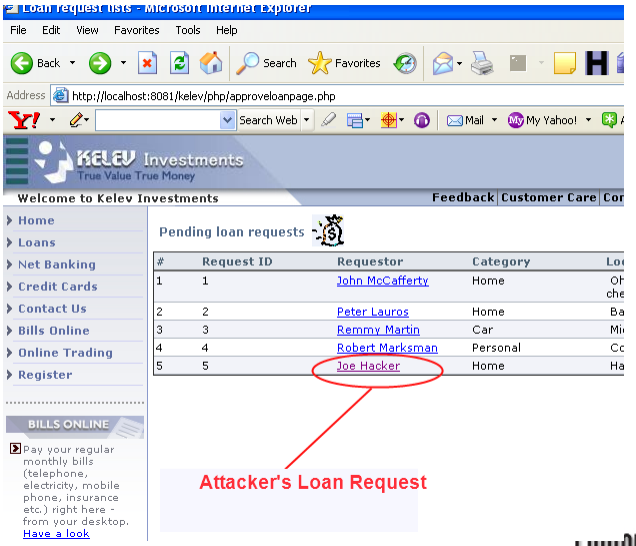
From: www.itsa.ufl.edu/2006/presentations/malpani.ppt

Cross Site Scripting – Stored XSS



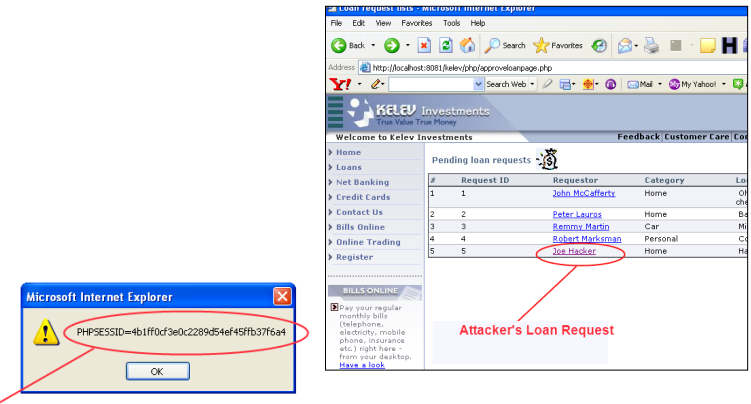
From: www.itsa.ufl.edu/2006/presentations/malpani.ppt

Cross Site Scripting – Stored XSS



From: www.itsa.ufl.edu/2006/presentations/malpani.ppt

Cross Site Scripting – Stored XSS



Unvalidated Input resulted in a Cross-Site Scripting Attack and the theft of the Administrator's Cookie

From: www.itsa.ufl.edu/2006/presentations/malpani.ppt

Computer Science
NC STATE UNIVERSITY

Mitigation

- Input filtering (blacklist/whitelist)
- Output filtering (blacklist/whitelist)
- Output encoding libraries
  - Microsoft's Anti-XSS library
  - OWASP ESAPI Encoding module
  - Encoding rules depend on context (HTML, HTML attribute, Script, URL query string, etc)

Computer Science
NC STATE UNIVERSITY

Testing for XSS

- Step1: Identify where untrusted input can be used as output
  - Welcome message, error message, etc.
- Step2: Test whether the input is not validated and valid HTML and script code can be executed

Computer Science
NC STATE UNIVERSITY

Testing for XSS

- Check if special characters are encoded

```
<XSS> vs. &ltXSS&gt
```

- Check if script can be executed

```
<script>alert("XSS")</script>
```

- Check if a double quote escape can be evaded

```
<script>alert(String.fromCharCode(88,83,
  83));<script>
```

Computer Science
NC STATE UNIVERSITY

http://ha.ckers.org/xss.html

Testing for XSS

- Check if input filtering can be evaded

```
<<SCRIPT>alert("XSS");//<</SCRIPT>
```

- Denial of service

```
<script>alert(document.cookie);<script>ar
ticle.php?title=<meta%20http-
equiv="refresh"%20content="0;">
```

http://ha.ckers.org/xss.html
http://www.owasp.org/index.php/Testing_for_Cross_site_scripting

Computer Science
NC STATE UNIVERSITY

Videos

- http://www.youtube.com/watch?v=r79ozjCL7DA

Computer Science
NC STATE UNIVERSITY

Cross site request forgery

12	CWE-352: Cross-Site Request Forgery (CSRF)

Summary			
Weakness Prevalence	High	Consequences	Data loss, Code execution
Remediation Cost	High	Ease of Detection	Moderate
Attack Frequency	Often	Attacker Awareness	Medium

- Attacker tricks a browser into performing undesired requests to websites on behalf of logged-in users
- The attack is performed by including in a page either an image or an iframe pointing to a site where the user is supposed to be logged in

Computer Science NC STATE UNIVERSITY

Video

- http://www.youtube.com/watch?v=uycmHQM_h64

Computer Science NC STATE UNIVERSITY

Successful CSRF

- Several things have to happen for cross-site request forgery to succeed:
  - The attacker must target either a site that doesn't check the referrer header (which is common)
  - The attacker must find a form submission at the target site, or a URL that has side effects, that does something (e.g., transfers money, or changes the victim's e-mail address or password).
  - The attacker must determine the right values for all the form's or URL's inputs; if any of them are required to be secret authentication values or IDs that the attacker can't guess, the attack will fail.
  - The attacker must lure the victim to a Web page with malicious code while the victim is logged in to the target site.

Computer Science
NC STATE UNIVERSITY

http://en.wikipedia.org/wiki/Cross-site_request_forgery

CSRF Mitigation

- Anti-CSRF token(s)
- Checking HTTP Referer or HTTP Origin header

Computer Science
NC STATE UNIVERSITY

WebScarab

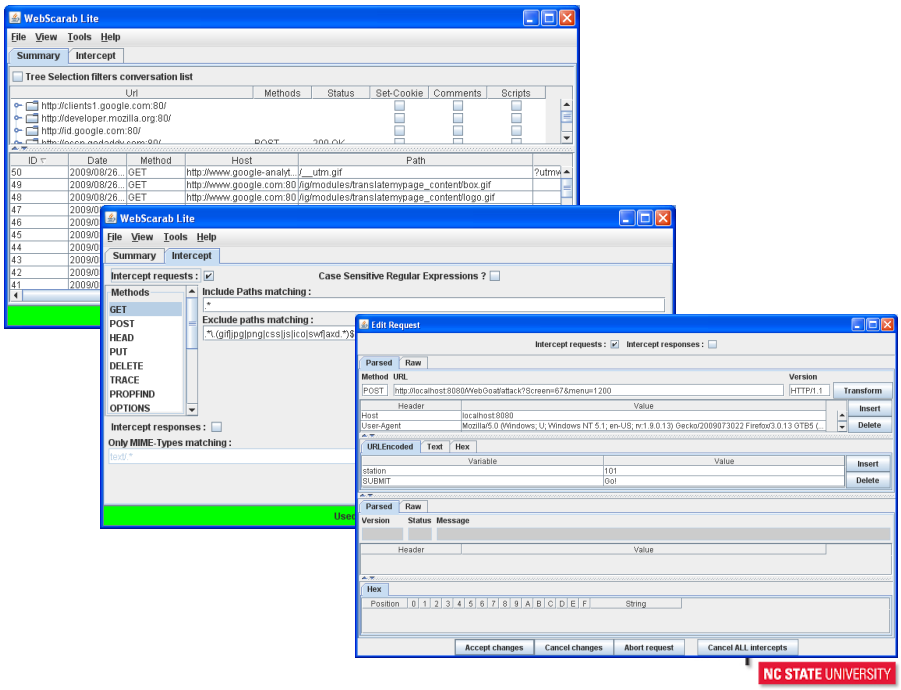
- A proxy to intercept HTTP requests and responses
- Simulate man-in-the-middle attack
- Can evade client-side input validation
- Usage
  1. Modify the manual HTTP proxy configuration on the browser to localhost:8008

     - Firefox: Tools -> Options -> Advanced -> Network -> Connection -> Settings

     - IE: Tools -> Internet Options -> Connections -> LAN Settings -> Proxy Settings

     (http://localhost.:8080/WebGoat/attack)
  2. Start WebScarab
  3. Select HTTP request methods (GET,POST)
  4. Check the intercept requests/responses checkbox if necessary
  5. Modify the requests and accept changes

NC STATE UNIVERSITY



NC STATE UNIVERSITY