Web application security

COINS Summer School 2019

Jingyue Li (Bill)

Associate Prof.

Dept. Computer Science

NTNU

Goal of teaching

No more - «Penetrate & Patch»

Outline

- ➢ Typical Web app security risks and mitigations
- My studies related to Web app security

10 Most Critical Web Application Security Risks

OWASP Top 10 - 2013		OWASP Top 10 - 2017
A1 – Injection	→	A1:2017-Injection
A2 – Broken Authentication and Session Management	→	A2:2017-Broken Authentication
A3 – Cross-Site Scripting (XSS)	↘	A3:2017-Sensitive Data Exposure
A4 – Insecure Direct Object References [Merged+A7]	∪	A4:2017-XML External Entities (XXE) [NEW]
A5 – Security Misconfiguration	↘	A5:2017-Broken Access Control [Merged]
A6 – Sensitive Data Exposure	↗	A6:2017-Security Misconfiguration
A7 – Missing Function Level Access Contr [Merged+A4]	∪	A7:2017-Cross-Site Scripting (XSS)
A8 – Cross-Site Request Forgery (CSRF)	☒	A8:2017-Insecure Deserialization [NEW, Community]
A9 – Using Components with Known Vulnerabilities	→	A9:2017-Using Components with Known Vulnerabilities
A10 – Unvalidated Redirects and Forwards	☒	A10:2017-Insufficient Logging&Monitoring [NEW,Comm.]

Injection Attacks

Injection attacks

- SQL injection
- Blind SQL injection
- Xpath injection
- …

Injection attack

- Malicious inputs inserted into
  - Query/Data
  - Command
- Attack string alters intended semantics
  - Query/Data
  - Command

SQL injection – normal input

“Server side login code (E.g., PHP)”

$ result = mysql_query (“ select * from Users where (name = ‘$ user’ and password = ‘$pass’); ”);

Application constructs SQL query from parameter to DB, e.g.

Select * from
Users where name = user1 and password = OK123456

SQL injection – Attack scenario (1)

- Attacker types in this in ***username*** field

  user1 ’ OR 1=1); --

- At the server side, the code to be executed

  $ result = mysql_query (“ select * from Users where (name = ‘user1 ’ OR 1=1); -- and password = ‘whocares’); ”);

  - SQL query constructed is

  Select * from Users

  Where name = user1 OR 1= 1

1=1 is always true. All user data compromised

SQL injection – Attack scenario (2)

- If attacker types this in ***username*** field

  user1 ’ OR 1=1); Drop TABLE Users; --

- SQL query constructed is

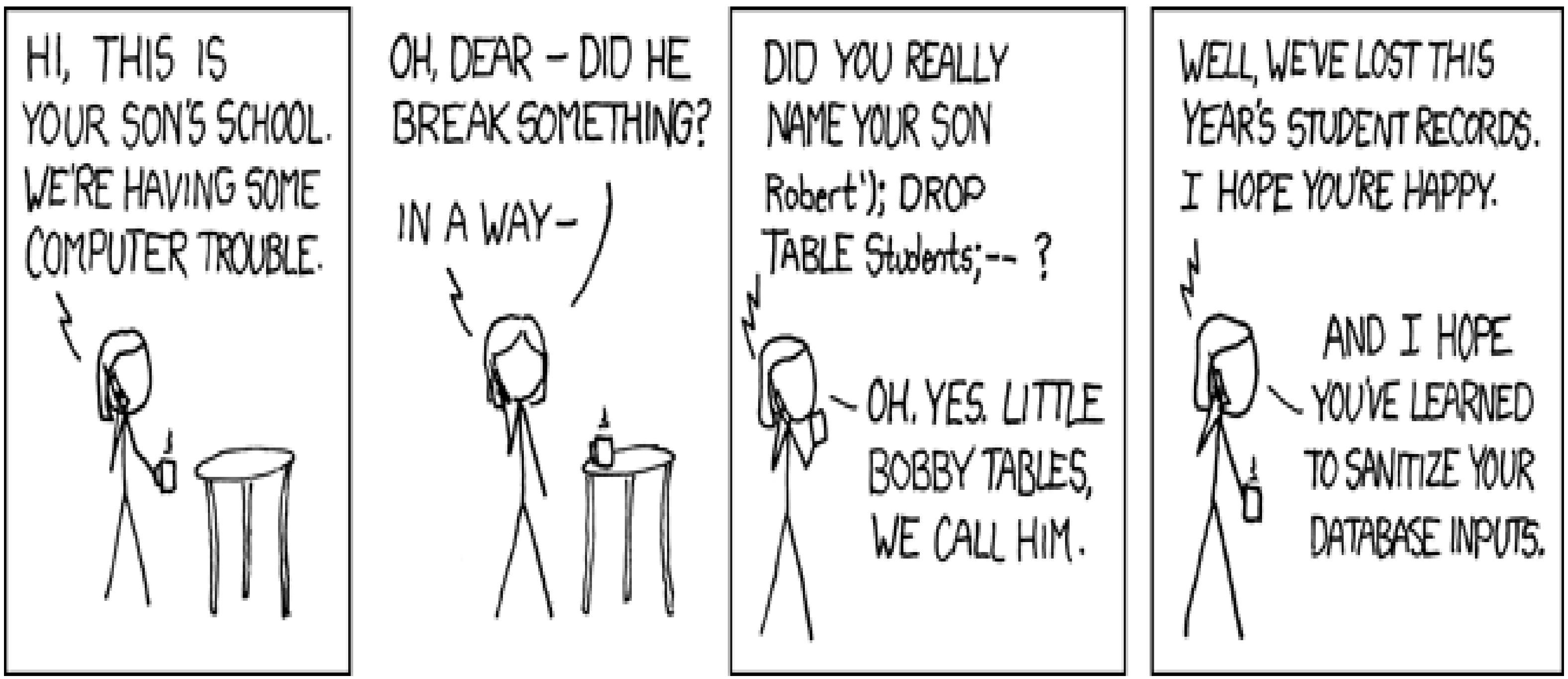
  Select * from Users

  Where name = user1 OR 1=1;

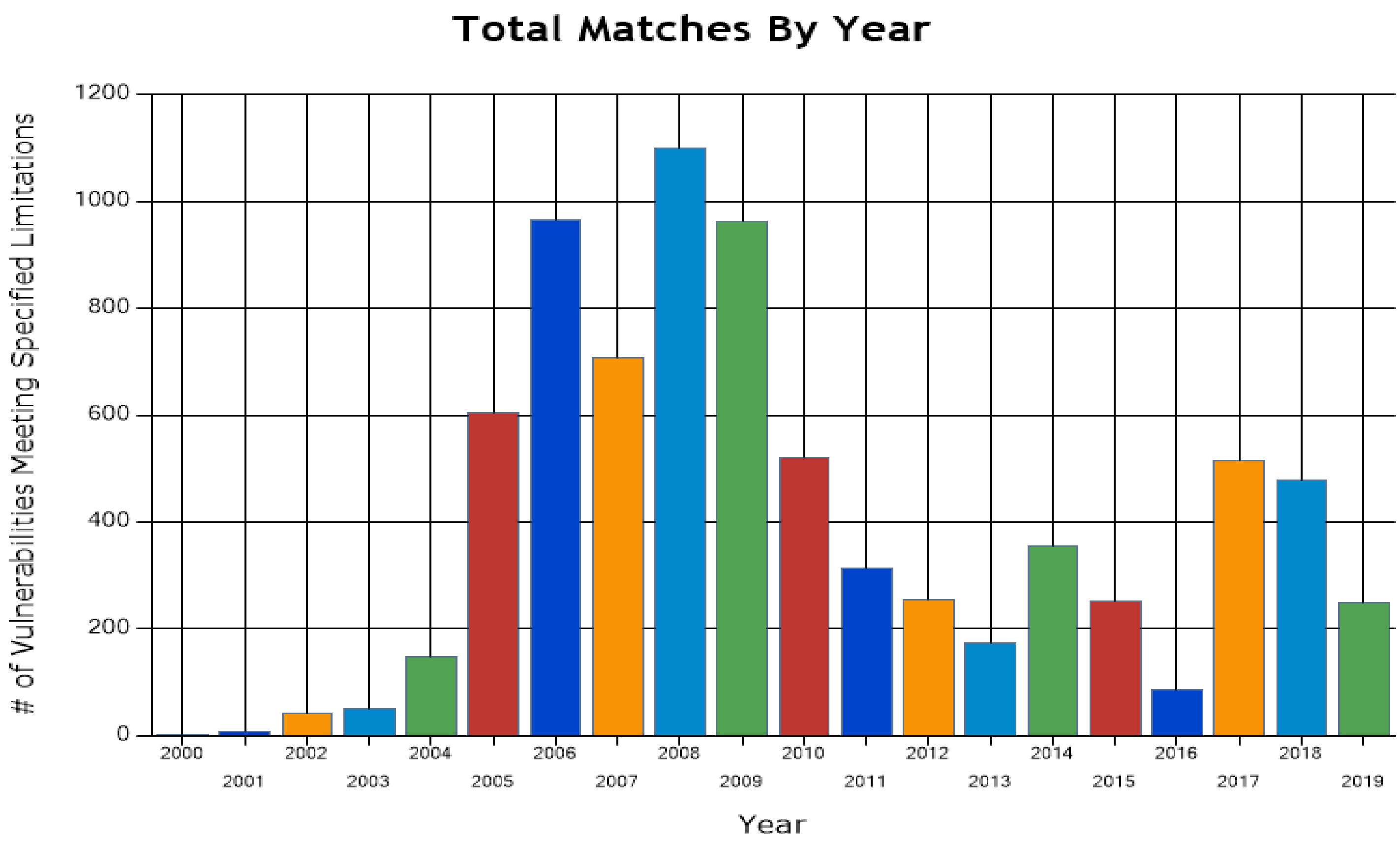
  Drop TABLE Users;

Delete the Table Users

SQL injection Humor



Is SQL injection just a humor?



By searching key word *SQL injection* in https://nvd.nist.gov/vuln/search/statistics?form_type=Basic&results_type=statistics&query=sql+injection&search_type=all

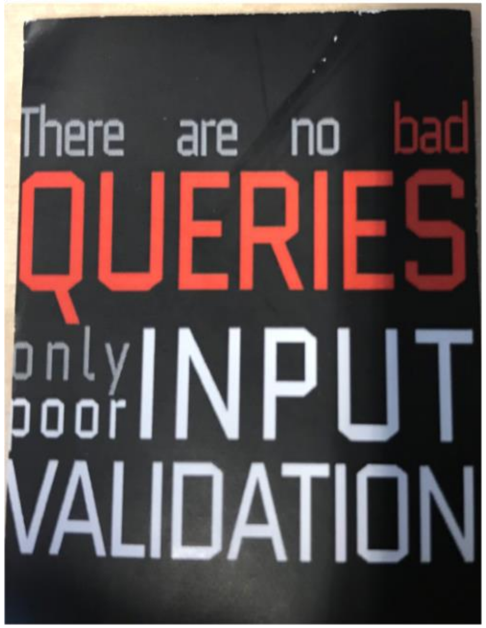
Why so common?

What can you achieve?

- Bypass authentication
- Privilege escalation
- Stealing information
- Destruction

SQL injection countermeasures

- Blacklisting
- Whitelisting
- Escaping
- Prepared statement & bind variables
- Mitigating impact



<< All input is evil. >> Michael Howard

Blacklisting

- Filter quotes, semicolons, whitespace, and …?
  - E.g. Kill_quotes (Java) removes single quotes

```
String kill_quotes(String str) {
  StringBuffer result = new    StringBuffer(str.length());
  for (int i = 0; i < str.length(); i++) {
    if (str.charAt(i) != '\'')
      result.append(str.charAt(i));
  }
  return result.toString();
}
```

user1 ’ OR 1=1); --

Pitfalls of Blacklisting

- Could always miss a dangerous character

- May conflict with functional requirements
  - E.g. A user with name O’Brien

Whitelisting

- Only allow well-defined safe inputs
- Using RegExp (regular expressions) match string
  - E.g. *month* parameter: non-negative integer
    - RegExp: ^[0-9]+$
    - ^ beginning of string, $ end of string
    - [0-9] + matches a digit, + specifies 1 or more
- Pitfalls: Hard to define RegExp for all safe values

Escaping

- Could escape quotes instead of blacklisting
  - E.g. Escape(O’Brien) = O’’Brien

INSERT INTO USERS(username, passwd) VALUES (‘O’’Brien’, ‘mypasswd’)

- Pitfalls: like blacklisting, could always miss a dangerous character

Prepared statements & Bind variables

- Root cause of SQL injection attack
  - Data interpreted as control, e.g., user1 ' OR 1=1); --,
- Idea: decouple query statement and data input

Examples of PHP prepared statement

- Prepare the statement with placeholders
  – $ ps = $ db->prepare('SELECT * FROM Users WHERE name = ? and password = ?');

Bind variable;
Data Placeholder

- Specify data to be filled in for the placeholders
  – $ ps -> execute (array($current_username, $current_passwd));

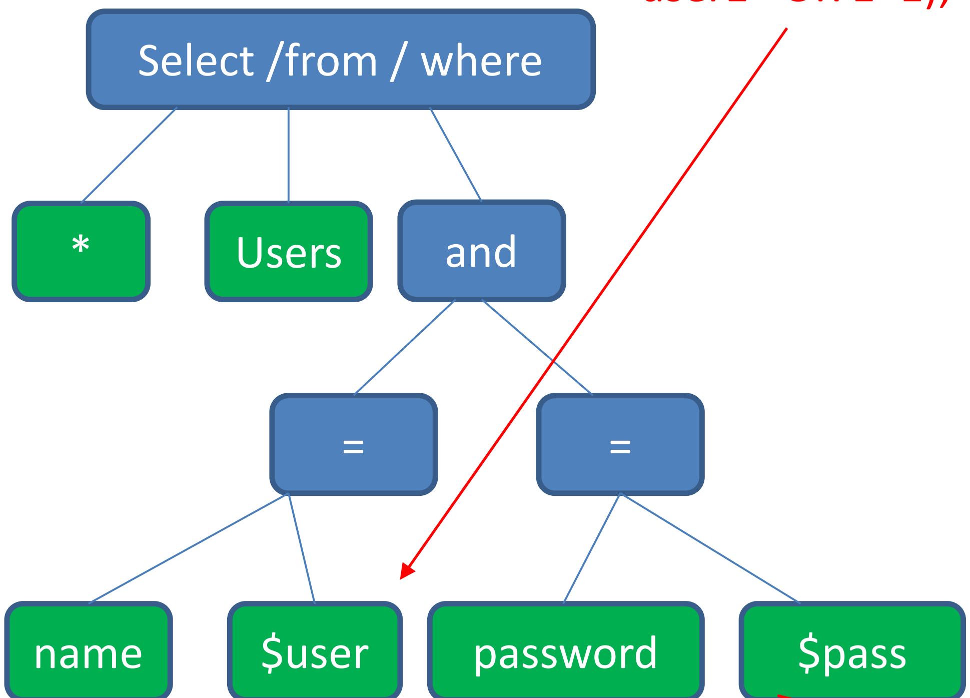
Why prepared statements & bind variables work?

- Decoupling lets us compile the prepared statement before binding the “query input data”
  - Prepared statements
    - Preserve the structure of intended query
    - “Query input data” is not involved in query parsing or compiling
  - Bind variables
    - ? Placeholders guaranteed to be data (not control)

Why Prepared statements & Bind variables work (cont')?

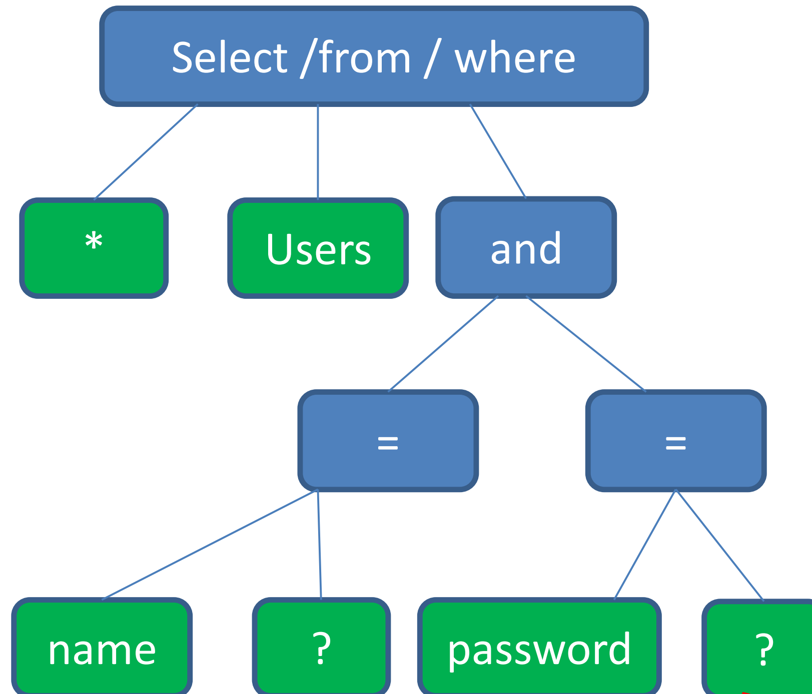
select * from Users where (name = '$user' and password = '$pass');

user1 ' OR 1=1); --



Malicious inputs can be interpreted as command during compiling

select * from Users where (name = '?' and password = '?');



Malicious inputs will always be interpreted as data during compiling

Mitigating impact

- Prevent schema & information leakage
  - E.g. Not display detailed error message to external users
  - E.g. Not display stack traces to external users
- Limiting privileges
  - No more privileges than typical user needs
  - E.g. Read access, tables/views user can query
  - E.g. No drop table privilege for typical user

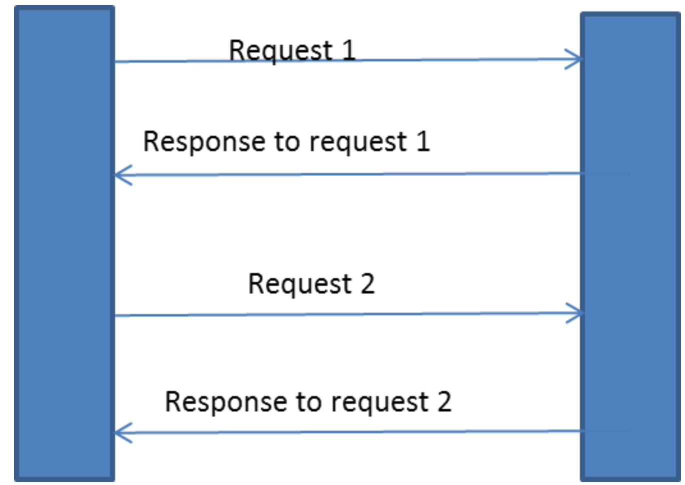
Mitigate impact (cont’)

- Encrypt sensitive data, e.g.,
  - Username, password, credit card number
- Key management precautions
  - Do not store encryption key in DB

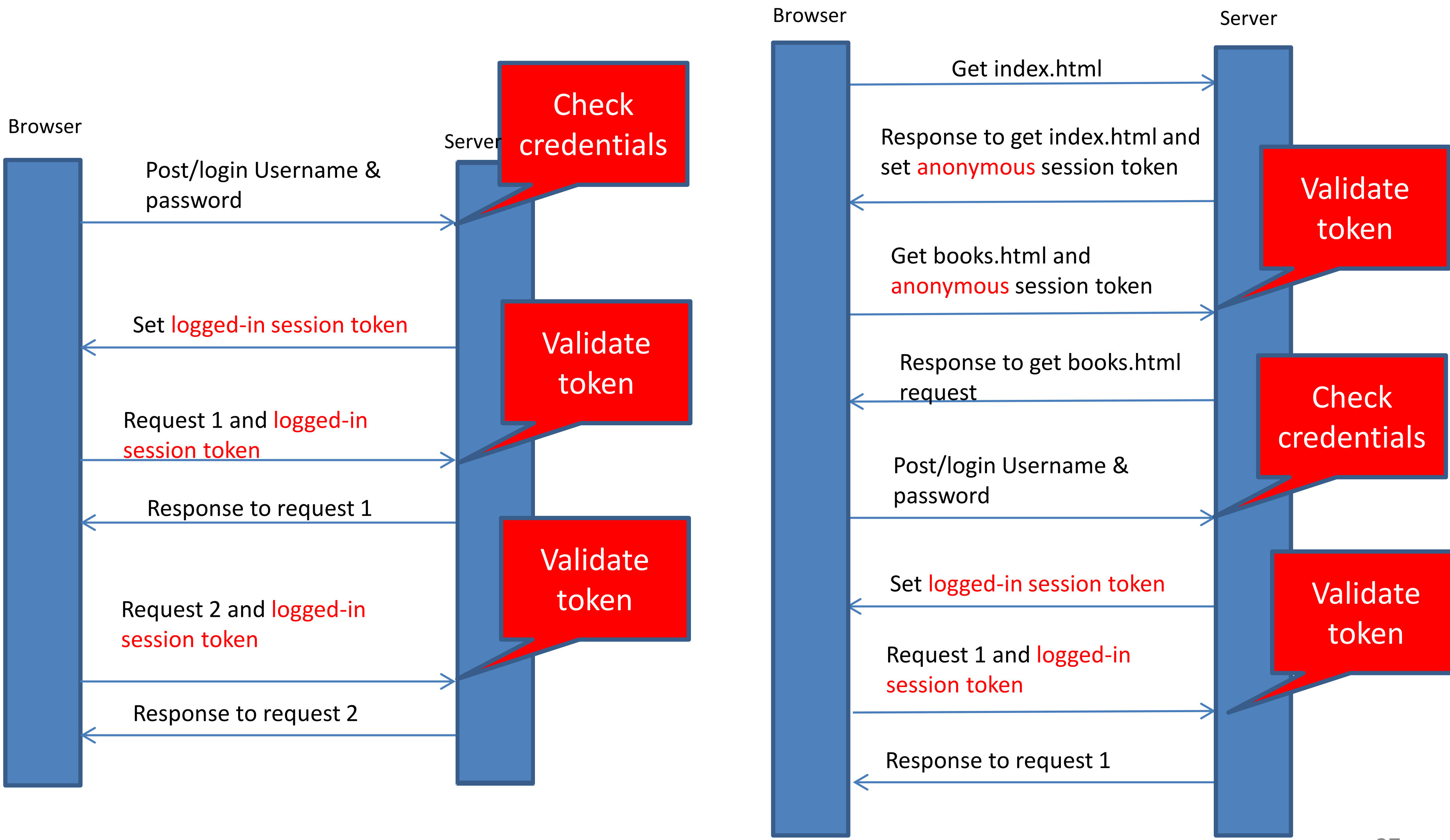
Session Management Attacks

Why session management?

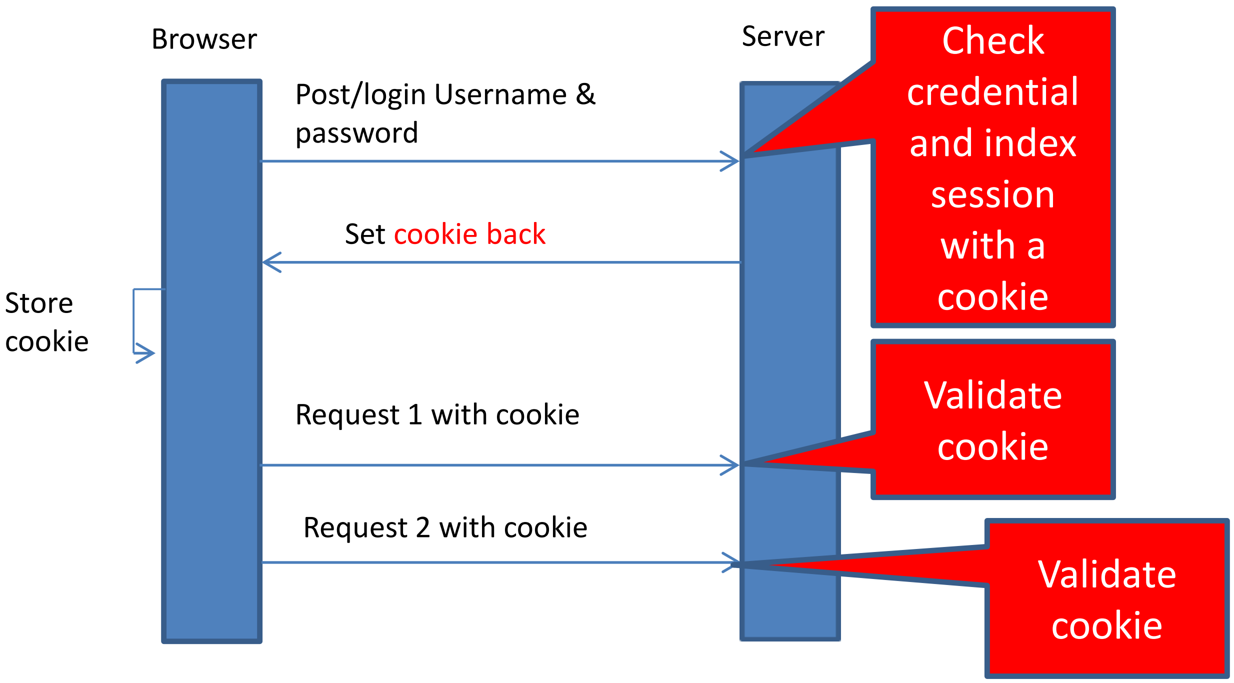
- HTTP is stateless
- Impossible to know if Req1 and Req2 are from same client
- Users would have to constantly re-authenticate
- Session management
  - Authenticate user once
  - All subsequent requests are tied to user



Session tokens



Session management with cookie



How cookie works

- Setting and sending cookies
  - In header of HTTP response (Server to browser)

    set-Cookie: token=**1234**; expire=Wed, 3-Aug-2016 08:00:00; path=/; domain = idi.ntnu.no

  - In header of HTTP request (Browser to server, when visit the domain of the same scope)

    Cookie: token=**1234**

- Cookie protocol problem
  - Sever only sees Cookie: NAME = VALUE
  - Server does not see which domain sends the cookie

Session management attacks and countermeasures

- Session token theft
- Session token predication attack
- Session fixation attack

Session token theft – Sniff network

- User
  - Alice logs in login.site.com (HTTPS)
  - Alice gets logged-in session token
  - Alice visits non-encrypted.site.com (HTTP)
- Attacker
  - Wait for Alice to login
  - Steal the logged-in session token (in HTTP)

    E.g. FireSheep (2010) sniff WiFi in wireless cafe
  - Impersonate Alice to issue request

Session token theft – Logout problem

- What should happen during logout
  - 1. Delete session token from client
  - 2. Mark session token as expired on server
  - Many web sites do (1) but not (2)!!
- Attacker
  - If can impersonate once, can impersonate for a long time
  - E.g. Twitter sad story
    - Token does not become invalid when user logs out

https://packetstormsecurity.com/files/119773/twitter-cookie.txt (2013)

Solutions to Session token theft

- Always send Session ID over encrypted channel
- Remember to log out
- Time out session ID
- Delete expired session ID
- Binding session token to client's IP or computer

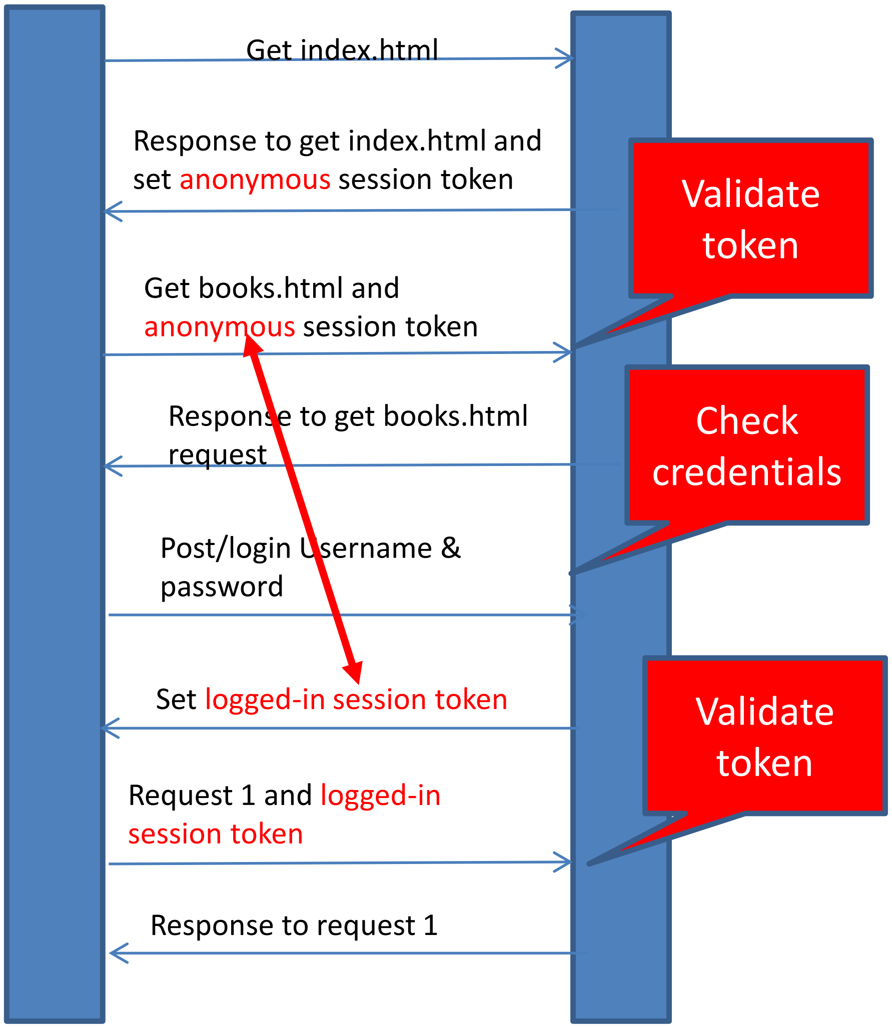
Binding session token to client's IP or Computer

- Idea:
  - Overcome cookie protocol problem
    - Sever only sees Cookie: NAME = VALUE
    - Server does not see which domain sends the cookie
- Combine IP
  - Possible issue: IP address changes (Wifi / 3G)
- Combine user agent: weak defense, but does not hurt

Session token predication attack

- Predicable tokens, e.g., counter
- Non-predicable token means
  - Seeing one or more token
  - Should not be able to predict other tokens
- Solution:
  - Do not invent own token generator algorithm
  - Use token generator from known framework (e.g., ASP, Tomcat, Rails)

Session fixation attack



- User
  - Visits site using anonymous token
- Attacker
  - Overwrites user's anonymous token with own token
- User:
  - Logs in and gets anonymous token elevated to logged-in token
- Attacker:
  - Attacker's token gets elevated to logged-in token after user logs in
- Vulnerability: Sever elevates the anonymous token without changing the value

How to overwrite session token?

- Tampering through network
  - Alice visits non-encrypted.site.com (HTTP)
  - Attacker injects into response to overwrite secure cookie

    Set-cookie: SSID=maliciousToken;
- Cross-site scripting
  - How?

Mitigate session fixation

- Always issue a new session token, when elevate from anonymous token to logged in token

Cross-Site Scripting (XSS) Attack

An application vulnerable to XSS

Client

Name:

Bob

Query

Query.html

http://example.com/query?name = Bob

Return result.html as response

Query string (i.e. Bob)
ECHOed back in result.html

Server

Render: result.html

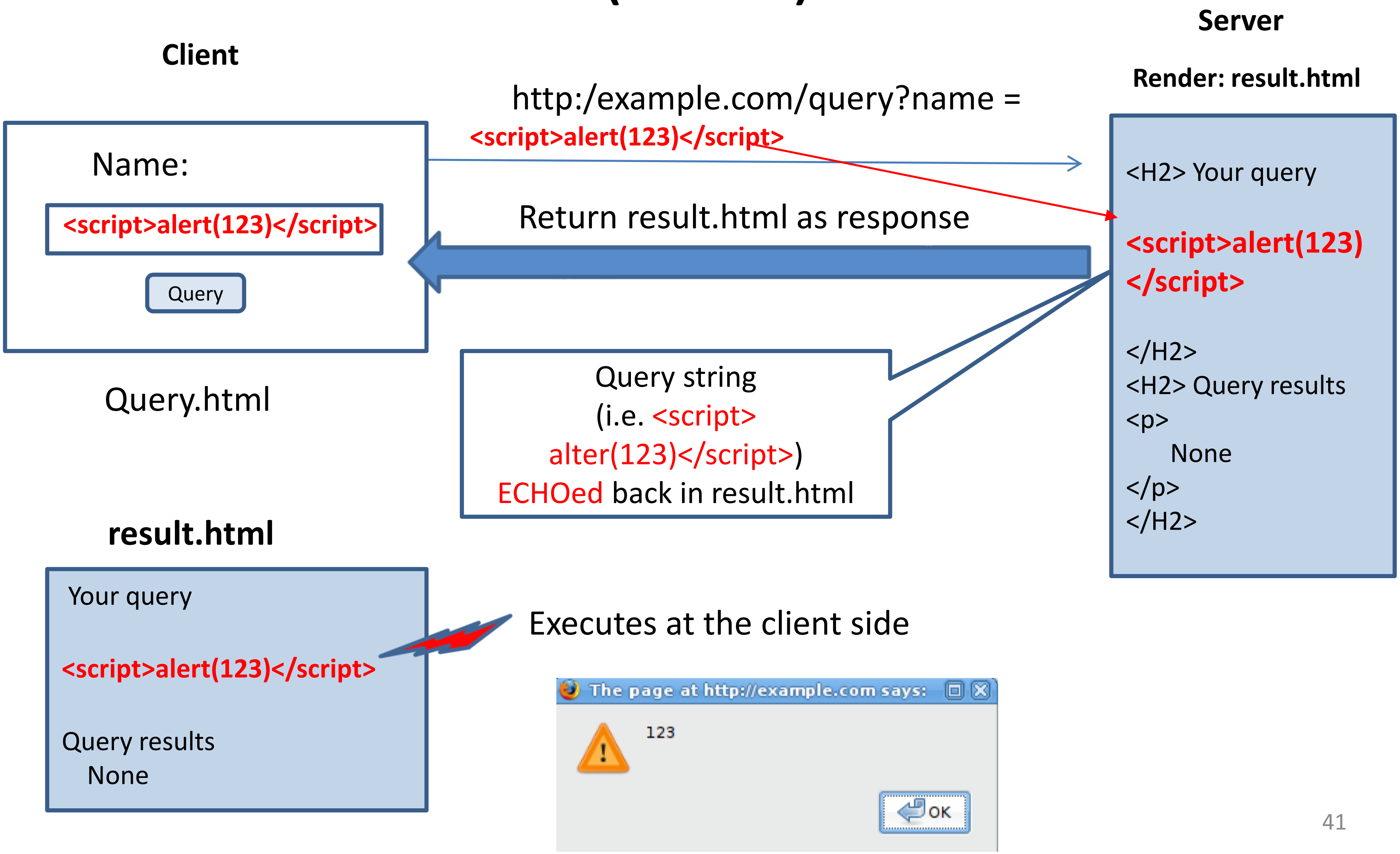
<H2> Your query
Bob
</H2>
<H2> Query results
<p>
Name: Bob Johnson
Tel: 123456
</p>
</H2>

result.html

Your query
Bob

Query results
Name: Bob Johnson
Tel: 123456

An application vulnerable to XSS (cont')



Session token overwritten using XSS

- Attacker
  - Find out http://example.com/query? is vulnerable to XSS
  - Get a valid anonymous token from the example.com, e.g., exampleComToken=1234
  - Send this link to user

    http://example.com/query?name = <script>

    document.cookie = ‘exampleComToken = 1234'

    </script>
  - Lure user to click this link
- User
  - Lured, clicks the link
  - The browser executes the script **document.cookie = ‘exampleComToken = 1234’** Overwrite user’s cookie value with attacker’s cookie value, i.e., 1234

XSS exploits

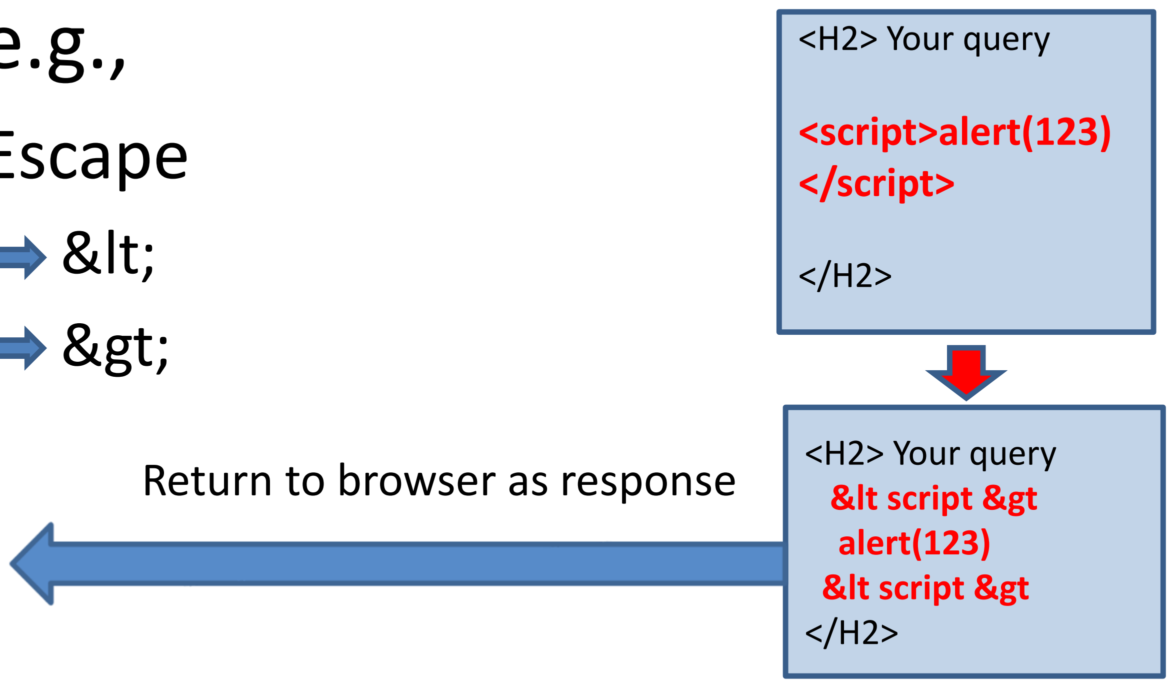
- Not just cookie theft/overwritten
- Attacker injects malicious script in your page
- Browser thinks it is your legitimate script
- Typical sources of untrusted input
  - Query
  - User/profile page (first name, address, etc.)
  - Forum/message board
  - Blog
  - Etc.

Reflected vs. Stored XSS

- Reflected XSS
  - Script injected into a request
  - Reflected immediately in response
- Stored XSS
  - Script injected into a request
  - Script stored somewhere (i.e., DB) in server
  - Reflected repeatedly
  - More easily spread

XSS mitigation

- Sanitize input data
- Sanitize / escape data inserted in web page
- Escape, e.g.,
  - HTML Escape
    - < ⟹ <
    - > ⟹ >



XML External Entities (XXE) Attack

XML External Entities

- Also called **EXTERNAL (PARSED) GENERAL ENTITY***
- They refer to data that an XML processor has to parse
- Useful for creating a common reference that can be shared between multiple documents

<!ENTITY name SYSTEM "URI">

External entity declaration

Private/local

Location

* http://xmlwriter.net/xml_guide/entity_declaration.shtml

XML External Entities Attack

- Against an application that parses XML input
- Untrusted **XML input containing a reference to an external entity is processed by a weakly configured XML parser**
- **Normal input**
  - **Input: <test> hello</test>**
  - **Output after XML parsing: hello**
- **Malicious input**
  - **Input: <!DOCTYPE test [!ENTITY xxefile SYSTEM "file:///etc/passwd">]><test> &xxefile </test>**
  - **Output: the content of file:///etc/passwd (SENSITIVE INFORMATION DISCLOSED)**

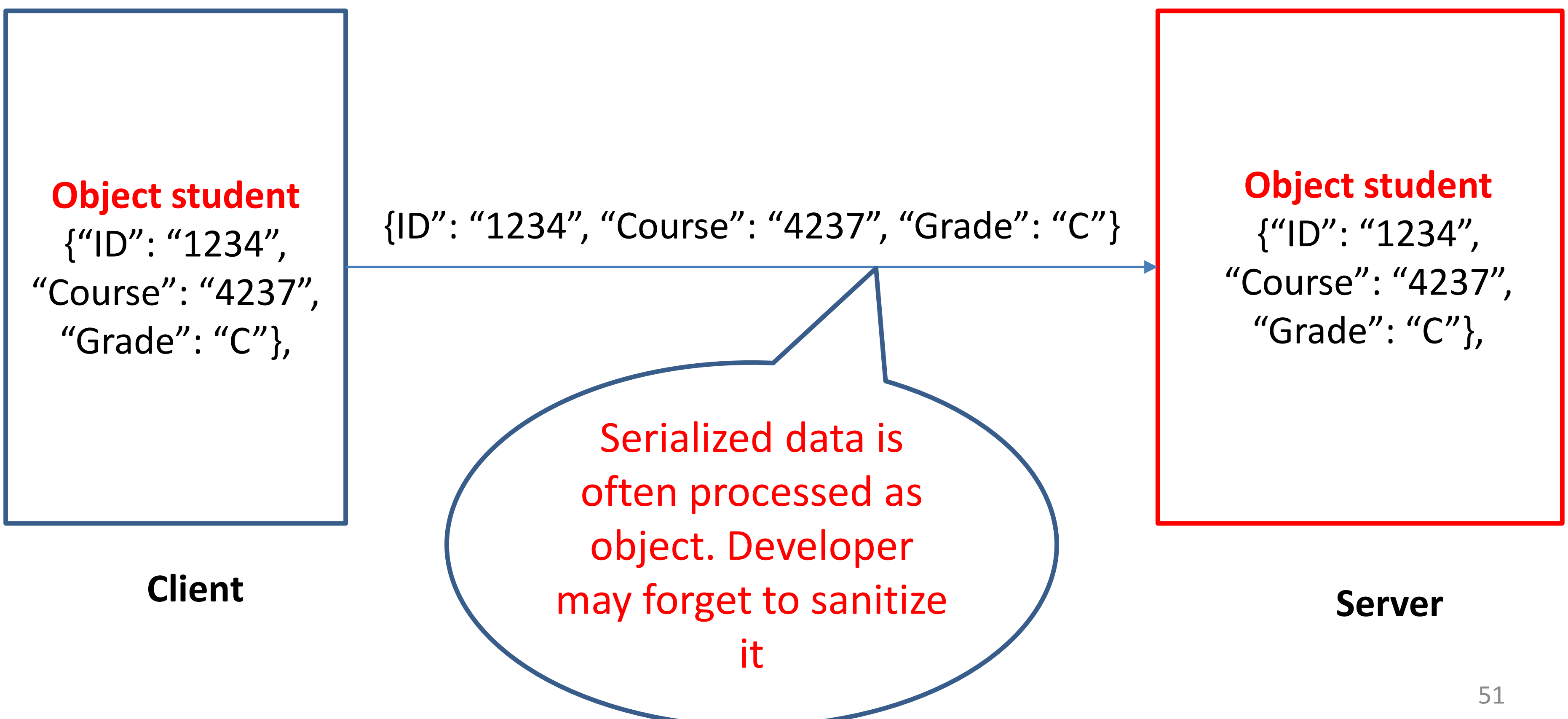
XML External Entities Countermeasure

- Disable XML external entity and DTD processing
- Input sanitization
  - Whitelisting
  - Web Application Firewalls

Insecure Deserialization Attack

Insecure Deserialization

- Serialization
- Deserialization



Insecure Deserialization Attack

- SQL injection
- Server side code
  - "SELECT Grade FROM student WHERE user = '"+ student.ID +"'; "
- Attacker
  - Tamper network data and inject SQL injection payload in serialized data stream

{"ID": " 'or'1'='1 ", "Course": "4237", "Grade": "C"}

- Developer does not sanitize serialized data. Then server will deserialize the data and use it to formulate object
  - "SELECT Grade FROM student WHERE user = 'or '1 = '1'; "

Insecure Deserialization Countermeasure

- Not to accept serialized objects from untrusted sources
- Implementing integrity checks such as digital signatures on any serialized objects
- Isolating and running code that deserializes in low privilege environments
- …

Insufficient Logging and Monitoring

Insufficient Logging and Monitoring

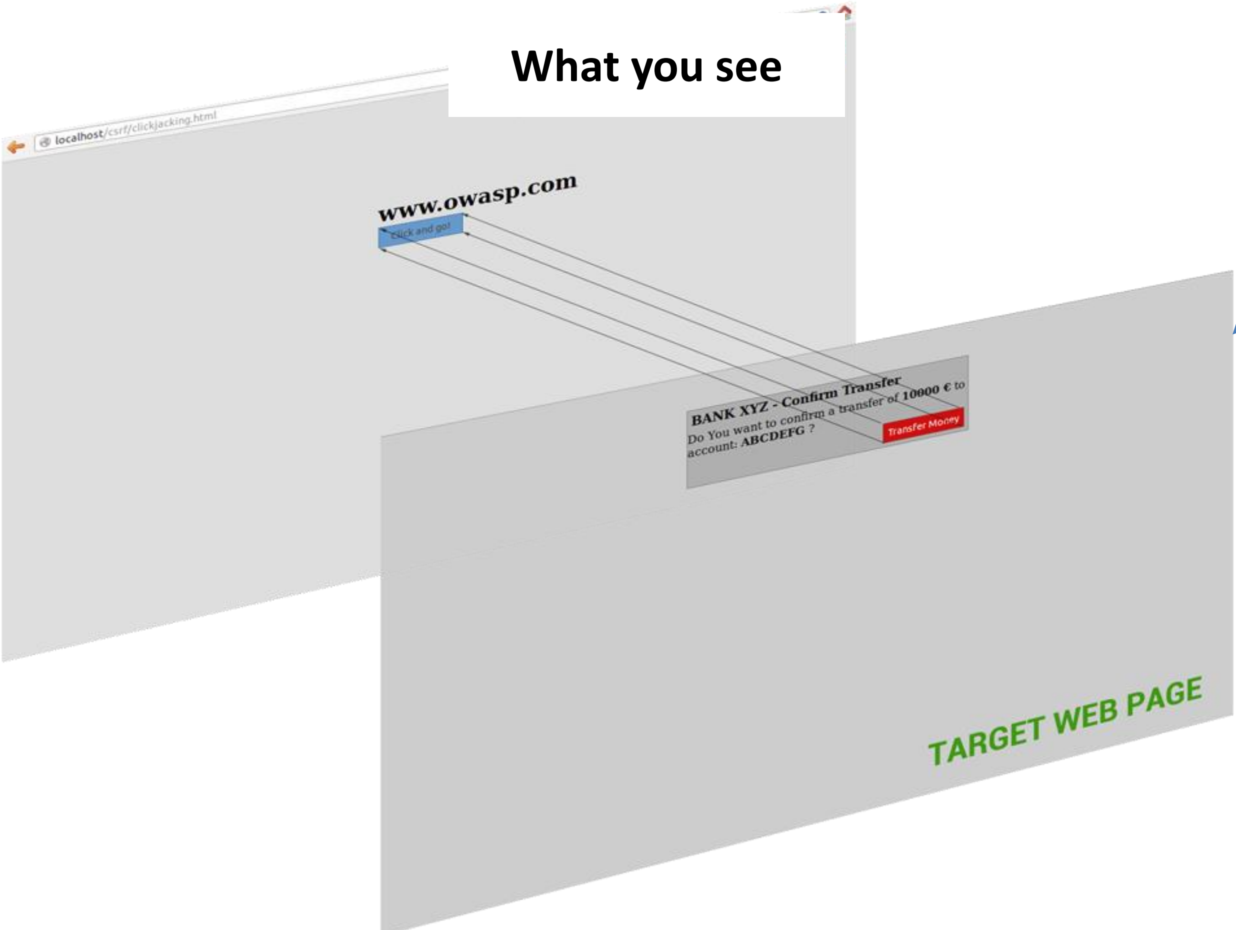
- Vulnerability
  - Auditable events, such as logins, failed logins, and high-value transactions are not logged
  - Warnings and errors generate no, inadequate, or unclear log messages
  - Logs of applications and APIs are not monitored for suspicious activity
  - Logs are only stored locally
  - Appropriate alerting thresholds and response escalation processes are not in place or effective
  - Unable to detect, escalate, or alert for active attacks in real time or near real time.

Insufficient Logging and Monitoring Countermeasure

- Ensure all login, access control failures, and server-side input validation failures can be logged with sufficient user context to identify suspicious or malicious accounts, and held for sufficient time to allow delayed forensic analysis
- Establish effective monitoring and alerting such that suspicious activities are detected and responded to in a timely fashion

HTML Attacks, e.g., Clickjacking Attack

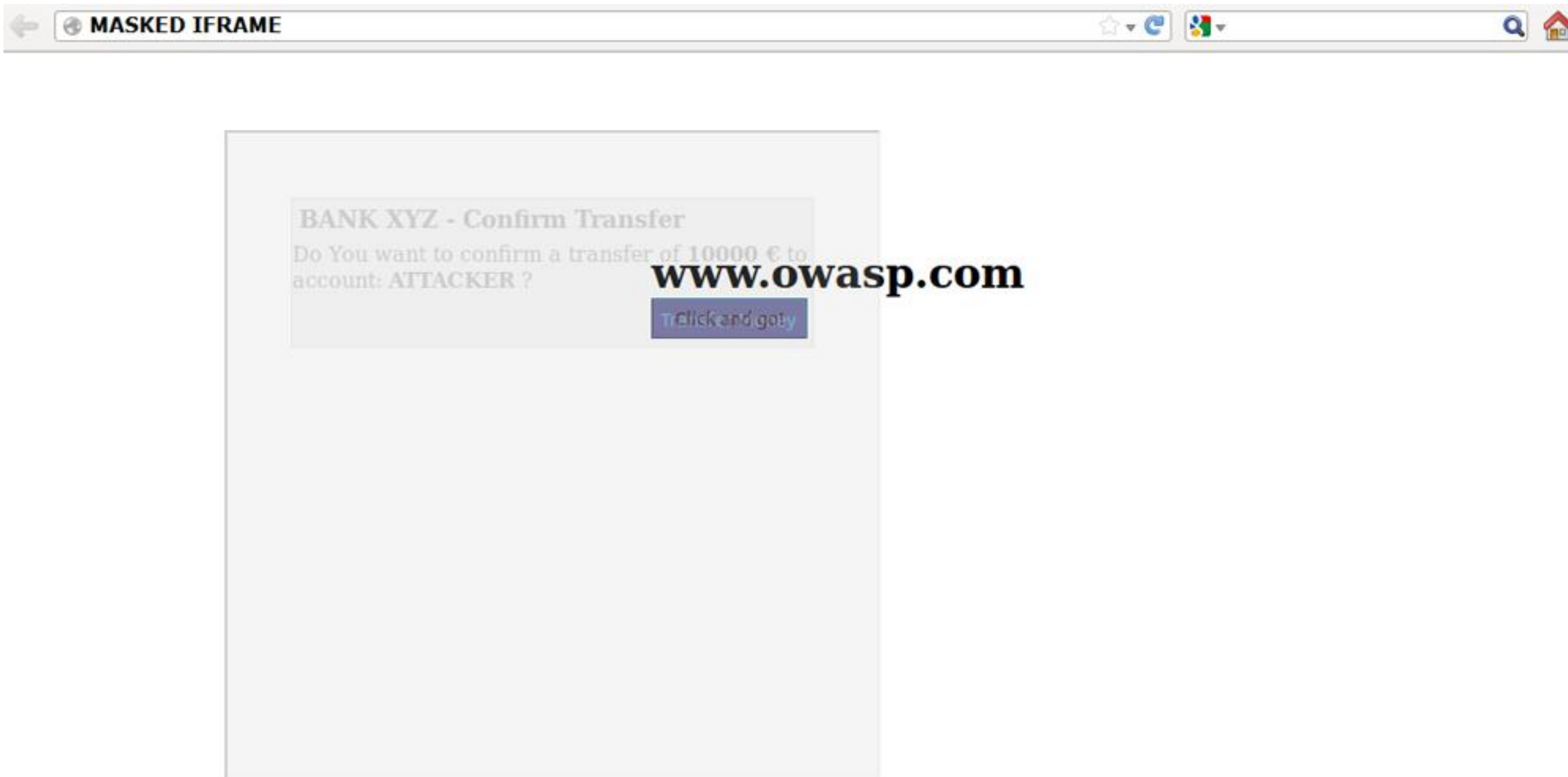
Clickjacking Attack



What you actually click is this page, but you cannot see this page, because it is transparent

Attacker overlays transparent frames to trick user into clicking on a button of another page, which contains malicious behavior

Clickjacking Attack (Cont')



Once the victim is surfing on the fictitious web page, he thinks that he is interacting with the visible user interface, but effectively he is performing actions on the hidden page.

HTML feature the clickjacking attacker exploits

- iframe and opacity

```
<html>
<head><title></title></head>
<body>

<iframe id= “top” src= “ http://attacker_wants_you_to_click_page.html” width = “1000” height = “3000”>
<iframe id=“bottom” src = “ http://attacker_wants_you__to_see_page.html ” width = “1000” height = “3000”>
```

Transparent

```
<style type = “text/css”>
 #top {position : absolute; top: 0px; left: 0px; opacity: 0.0}
 #bottom {position: absolute; top:0px; left: 0px; opacity: 1.0}

</body>
</html>
```

Defend against Clickjacking Attack

- Preventing other web pages from framing the site you want to defend (e.g., Defending with X-Frame-Options Response Headers)
- My site will not show in the frame, so that nobody can use my site to fool victim

If Facebook set “X-Frame-Options: deny”, Facebook will not show in <iFrame>

```
<html>
<head><title></title></head>
<body>
    <iframe id="bottom" src="https://www.facebook.com/" width="1000“ height="3000">
<style type ="text/css">
    #bottom {position: absolute; top:0px; left: 0px; opacity: 1.0}
</body>
</html>
```

Outline

- Typical Web app security risks and mitigations

➢ My studies related to Web app security

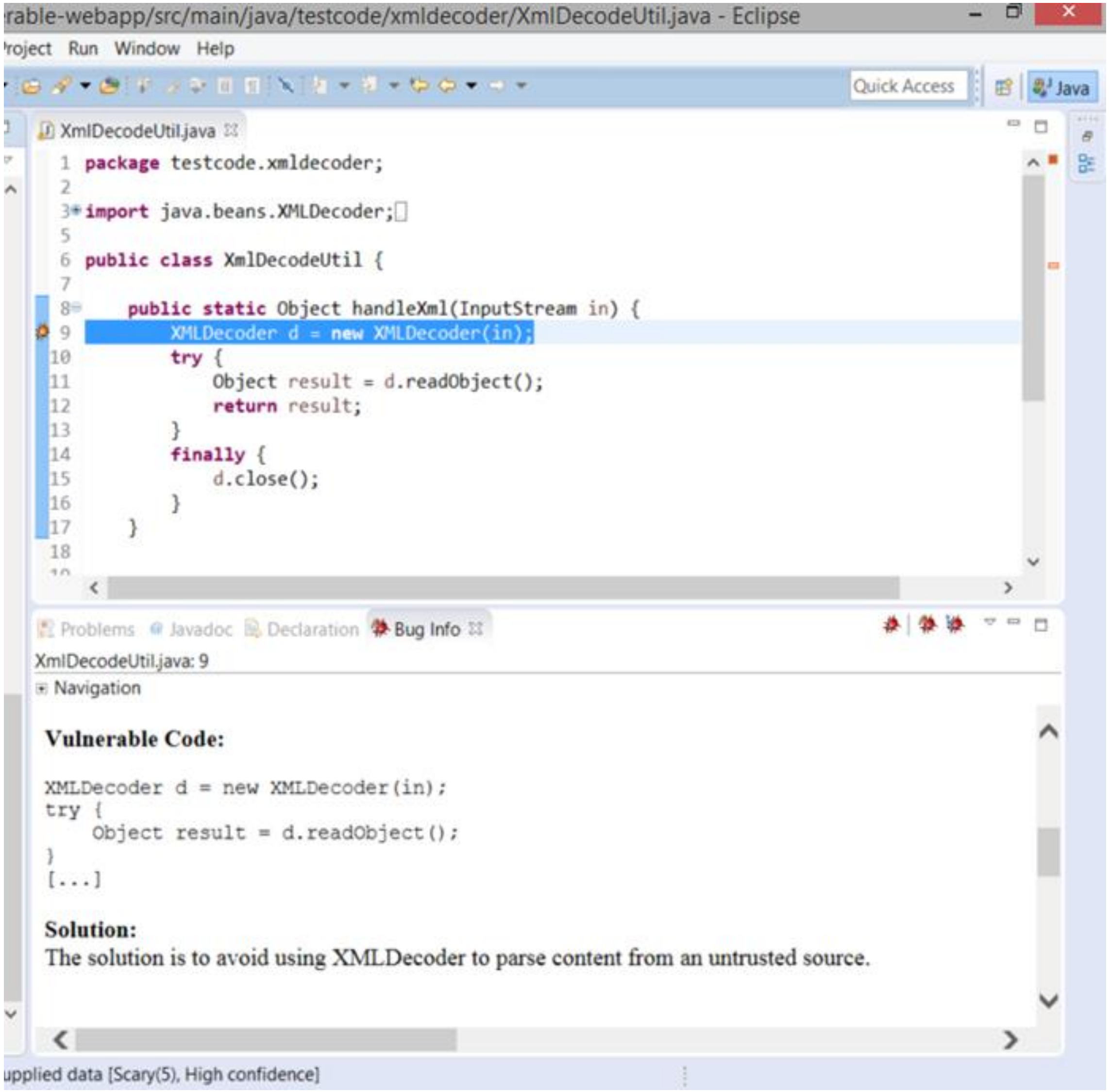
Study 1

- Evaluation of open-source IDE plugins for detecting Web application security vulnerabilities
- Research questions
  - RQ1: What is the coverage?
  - RQ2: How good is the performance?
  - RQ3: How good is the usability?

The paper is published at EASE (Evaluation and Assessment of Software Engineering) conference 2019.

IDE Plugins We Evaluate

- **ASIDE**
- **ESVD**
- **LAPSE+**
- **SpotBugs**
- **FindSecBugs**

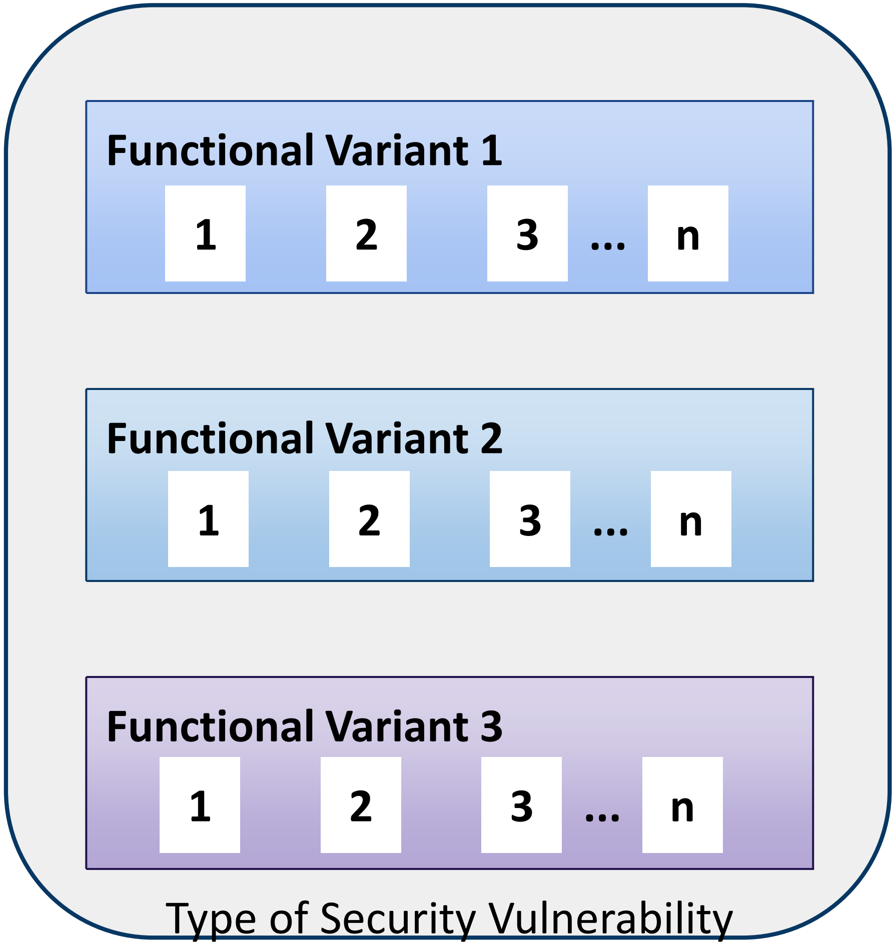


Vulnerable Code We Use for Evaluation

Juliet Test Suite v1.3

28,000 Test Cases

112 Security Vulnerabilities (CWE Entries)



Vulnerabilities and the test cases of the Juliet Test Suite

CWE		
ID	**Name**	
A1 Injection		**Total**
78	OS Command Injection	444
89	SQL Injection	2220
90	LDAP Injection	444
113	HTTP Response Splitting	1332
134	Use of Externally-Controlled Format String	666
643	XPath Injection	444
A2 Broken Authentication		**Total**
256	Unprotected Storage of Credentials	37
259	Use of Hard-coded Password	111
321	Use of Hard-coded Cryptographic Key	37
523	Unprotected Transport of Credentials	17
549	Missing Password Field Masking	17
A3 Sensitive Data Exposure		**Total**
315	Cleartext Storage of Sensitive Information in a Cookie	37
319	Cleartext Transmission of Sensitive Information	370
325	Missing Required Cryptographic Step	34
327	Use of a Broken or Risky Cryptographic Algorithm	34
328	Reversible One-Way Hash	51
329	Not Using a Random IV with CBC Mode	17
614	Sensitive Cookie in HTTPS Session Without 'Secure' Attribute	17
759	Use of a One-Way Hash without a Salt	17
760	Use of a One-Way Hash with a Predictable Salt	17
A5 Broken Access Control		**Total**
23	Relative Path Traversal	444
36	Absolute Path Traversal	444
566	Auth. Bypass Through User-Controlled SQL Primary Key	37
A6 Security Misconfiguration		**Total**
395	NullPointerException Catch to Detect NULL Pointer Deference	17
396	Declaration of Catch for Generic Exception	34
397	Declaration of Throws for Generic Exception	4
A7 Cross-Site Scripting		**Total**
80	Basic XSS	666
81	Improper Neutralization of Script in an Error Message	333
83	Improper Neutralization of Script in Attributes in a Web Page	333

Result of RQ1: Coverage

CWE		IDE-Integrated Static Analysis Tools									
ID		ASIDE		ESVD		LAPSE+		SpotBugs		FindSecBugs	
A1 Injection	Total	TP	FP	TP	FP	TP	FP	TP	FP	TP	FP
78	444	185	0	49	0	444	624	-	-	378	50
89	2220	3	3	1440	2280	2220	3060	2220	3000	1900	300
90	444	185	0	0	0	0	0	-	-	379	50
113	1332	555	795	0	0	0	0	57	0	989	0
134	666	148	212	-	-	-	-	-	-	462	0
643	444	185	265	0	0	444	1248	-	-	379	49
A2 Broken Authentication	Total	TP	FP	TP	FP	TP	FP	TP	FP	TP	FP
256	37	-	-	-	-	-	-	-	-	-	-
259	111	-	-	-	-	-	-	15	0	48	0
321	37	-	-	-	-	-	-	-	-	16	0
523	17	-	-	-	-	-	-	-	-	-	-
549	17	-	-	-	-	-	-	-	-	-	-
A3 Sensitive Data Exposure	Total	TP	FP	TP	FP	TP	FP	TP	FP	TP	FP
315	37	-	-	-	-	-	-	-	-	0	0
319	370	-	-	-	-	-	-	-	-	259	369
325	34	-	-	-	-	-	-	-	-	-	-
327	34	-	-	-	-	-	-	-	-	17	0
328	51	-	-	-	-	-	-	-	-	51	0
329	17	-	-	-	-	-	-	-	-	17	0
614	17	-	-	-	-	-	-	-	-	16	0
759	17	-	-	-	-	-	-	-	-	-	-
760	17	-	-	-	-	-	-	-	-	-	-
A5 Broken Access Control	Total	TP	FP	TP	FP	TP	FP	TP	FP	TP	FP
23	444	108	0	0	0	444	624	19	0	378	52
36	444	108	0	0	0	444	624	16	0	378	49
566	37	36	0	-	-	37	0	-	-	-	-
A6 Security Misconfiguration	Total	TP	FP	TP	FP	TP	FP	TP	FP	TP	FP
395	17	-	-	0	0	-	-	-	-	-	-
396	34	-	-	0	0	-	-	-	-	-	-
397	4	-	-	0	0	-	-	-	-	-	-
A7 Cross-Site Scripting	Total	TP	FP	TP	FP	TP	FP	TP	FP	TP	FP
80	666	642	900	28	0	666	936	19	0	666	76
81	333	321	450	14	0	0	0	19	0	333	38
83	333	108	0	14	0	333	468	19	0	333	38

Claimed vs. Confirmed Coverage

Tools	Confirmed Coverage		Claimed Coverage	
ASIDE	12	41%	12	41%
ESVD	5	17%	13	45%
LAPSE+	8	28%	11	38%
SpotBugs	8	28%	8	28%
FindSecBugs	18	62%	19	66%

Result of RQ2: Performance

CWE		IDE-Integrated Static Analysis Tools									
ID		ASIDE		ESVD		LAPSE+		SpotBugs		FindSecBugs	
A1 Injection	Total	TP	FP	TP	FP	TP	FP	TP	FP	TP	FP
78	444	185	0	49	0	444	624	-	-	378	50
89	2220	3	3	1440	2280	2220	3060	2220	3000	1900	300
90	444	185	0	0	0	0	0	-	-	379	50
113	1332	555	795	0	0	0	0	57	0	989	0
134	666	148	212	-	-	-	-	-	-	462	0
643	444	185	265	0	0	444	1248	-	-	379	49
A2 Broken Authentication	Total	TP	FP	TP	FP	TP	FP	TP	FP	TP	FP
256	37	-	-	-	-	-	-	-	-	-	-
259	111	-	-	-	-	-	-	15	0	48	0
321	37	-	-	-	-	-	-	-	-	16	0
523	17	-	-	-	-	-	-	-	-	-	-
549	17	-	-	-	-	-	-	-	-	-	-
A3 Sensitive Data Exposure	Total	TP	FP	TP	FP	TP	FP	TP	FP	TP	FP
315	37	-	-	-	-	-	-	-	-	0	0
319	370	-	-	-	-	-	-	-	-	259	369
325	34	-	-	-	-	-	-	-	-	-	-
327	34	-	-	-	-	-	-	-	-	17	0
328	51	-	-	-	-	-	-	-	-	51	0
329	17	-	-	-	-	-	-	-	-	17	0
614	17	-	-	-	-	-	-	-	-	16	0
759	17	-	-	-	-	-	-	-	-	-	-
760	17	-	-	-	-	-	-	-	-	-	-
A5 Broken Access Control	Total	TP	FP	TP	FP	TP	FP	TP	FP	TP	FP
23	444	108	0	0	0	444	624	19	0	378	52
36	444	108	0	0	0	444	624	16	0	378	49
566	37	36	0	-	-	37	0	-	-	-	-
A6 Security Misconfiguration	Total	TP	FP	TP	FP	TP	FP	TP	FP	TP	FP
395	17	-	-	0	0	-	-	-	-	-	-
396	34	-	-	0	0	-	-	-	-	-	-
397	4	-	-	0	0	-	-	-	-	-	-
A7 Cross-Site Scripting	Total	TP	FP	TP	FP	TP	FP	TP	FP	TP	FP
80	666	642	900	28	0	666	936	19	0	666	76
81	333	321	450	14	0	0	0	19	0	333	38
83	333	108	0	14	0	333	468	19	0	333	38

A study* at Microsoft shows that "90% of the participants are willing to accept a 5% false positive rate, while 47% of developers accept up to a 15% false positive rate." of source code analysis tools.

	ASIDE	ESVD	LAPSE+	SpotBugs	FindSecBugs
Averaged false positive rate	29%	12%	53%	7%	9%

* Maria Christakis and Christian Bird. 2016. What developers want and need from program analysis: an empirical study. InProceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering

Result of RQ3: Usability

		ASIDE	ESVD	LAPSE+	SpotBugs	FindSecBugs
Detailed information	What is the problem	×	✓	✓	✓	✓
	Why is it a problem	N/A	×	×	✓	✓
	How to fix the problem	N/A	×	×	×	✓
	Prioritized output	×	✓	×	✓	✓
	Quick fixes	✓	✓	×	×	×
	(E)arly or (L)ate detection	E	E	L	E/L	E/L
	Can suppress warnings	✓	✓	×	×	×
	Eclipse Environment integration	✓	✓	✓	✓	✓
	Available on Eclipse Marketplace	×	×	×	✓	×
	(I)mmediate or (N)egotiated interruptions	N	N	N	N	N
	Easily extendable	×	×	×	✓	×
	Possible to analyze single file only	×	×	×	✓	✓
	Possible to analyze single method only	×	×	×	×	×

Study 2

- Understanding and improving the open-source IDE plugins for better performance
- Research questions
  - RQ1: How is the plugin implemented?
  - RQ2: Why is the performance poor?
  - RQ3: How to improve the performance?

Study design

- Read the doc. and source code of the plugins
- For each false positive and false negative result, investigate why it happens and generate hypotheses
- Improve the code and re-test to verify the hypotheses
- Focus only on ESVD, SpotBug, and FindSecBug

How are the test cases in Juliet Test Suite structured?

- Source variant

```
/* SOURCE VARIANT: Read data using an outbound tcp connection */
socket = new Socket("host.example.org", 39544);
reader = new InputStreamReader(socket.getInputStream(), "UTF-8");
readerBuffered = new BufferedReader(reader);
data = readerBuffered.readLine();
```

```
/* SOURCE VARIANT: Read data from a file */
file = new File("C:\\data.txt");
stream = new FileInputStream(file);
reader = new InputStreamReader(stream, "UTF-8");
readerBuffered = new BufferedReader(reader);
data = readerBuffered.readLine();
```

How are the test cases in Juliet Test Suite structured (cont')?

- Control flow variant, e.g.,

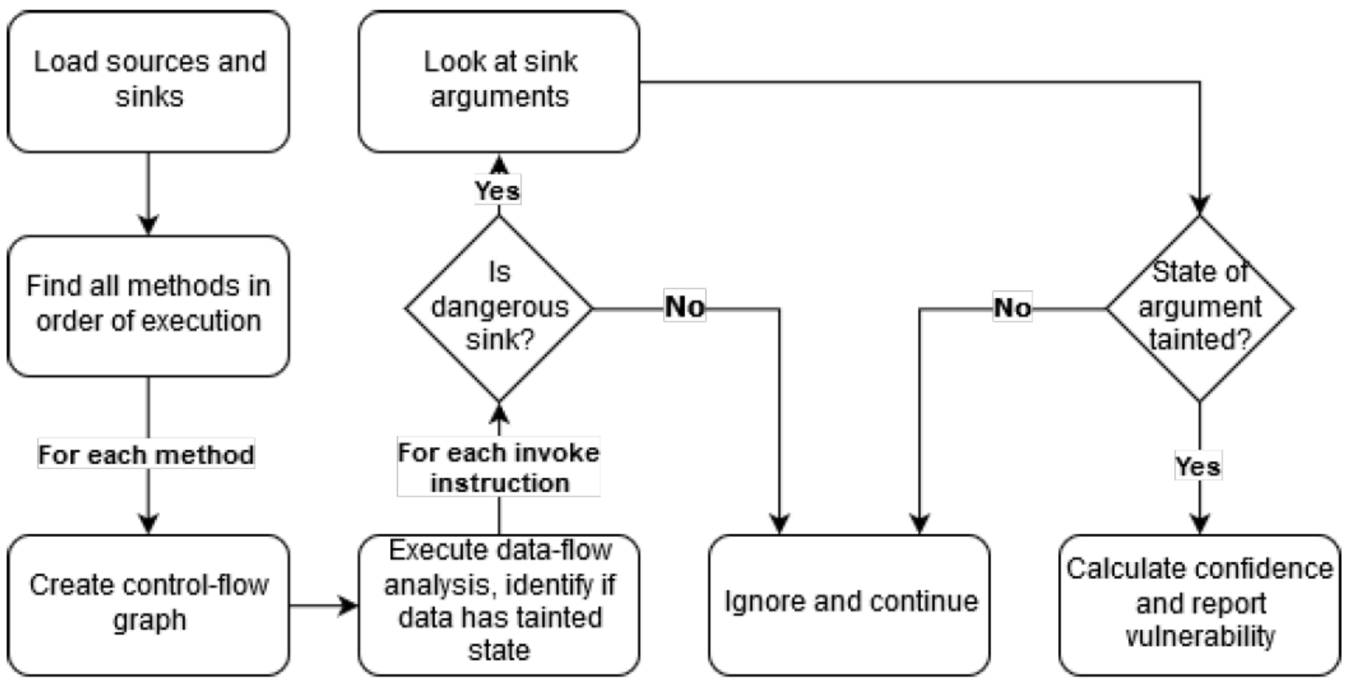
Flow Variant	Condition
02	The `boolean` value `true`.
03	The equation `5==5`.
04	A `private static final` constant set to the `boolean` value `true`.

- Data flow variant, e.g.,

Flow Variant	Description
31	Data is copied within the same method.
41	Data is passed as an argument from one method to another in the same class.
42	Data is returned from one method to another in the same class.

Result of RQ1: How is the plugin implemented?

- ESVD: Java source code, taint analysis
- SpotBug: Bytecode, taint analysis
- FindSecBug: Bytecode, taint analysis



Result of RQ2: Why poor performance?

- Missing sources and sinks, e.g.,
  - Only HttpServletRequest.getParameter(), HttpServletRequest.geteQueryString(), and HttpServletRequest.getHeader() are in sources defined in Spotbug, which lead to its bad recall of "*HTTP Response Splitting*" vulnerability
- Inadequate algorithm for analyzing control and data flow variants

Result of RQ2: Why poor performance (cont')?

- Bad principle and design, e.g.,
  - Spotbug and ESVD report all concatenated string variables as SQL injection vulnerabilities, which leads to high false positive.
- Uncertain detections are still reported, which leads to high false positive
- We also find limitations of the Julie Test Suite

Result of RQ3: How to improve performance?

- After proof-of-concept improvements

Injection		ESVD			SporBug			FindSeBug		
		Rec.	Prec.	Disc.	Rec.	Prec.	Disc.	Rec.	Prec.	Disc.
CWE-78 OS Cmd Inj.	RQ2	11%	100%	11%	0%	0%	0%	86%	86%	72%
	RQ3	19%	100%	19%	0%	0%	0%	89%	100%	86%
CWE-89 SQL Inj.	RQ2	65%	39%	0%	100%	43%	0%	86%	86%	72%
	RQ3	22%	100%	22%	84%	70%	49%	89%	100%	86%
CWE-90 LDAP Inj.	RQ2	0%	0%	0%	0%	0%	0%	86%	86%	72%
	RQ3	19%	100%	19%	0%	0%	0%	89%	100%	86%
CWE-113 HTTP R.S.	RQ2	0%	0%	0%	4%	100%	4%	74%	100%	74%
	RQ3	19%	100%	19%	47%	100%	47%	89%	100%	86%
CWE-643 XPath Inj.	RQ2	0%	0%	0%	0%	0%	0%	86%	86%	72%
	RQ3	0%	0%	0%	0%	0%	0%	89%	100%	86%

Result of RQ3: How to improve performance (cont')?

- After proof-of-concept improvements

		ESVD			SporBug			FindSeBug		
Broken Access Control - Path Traversal										
		Rec.	**Prec.**	**Disc.**	**Rec.**	**Prec.**	**Disc.**	**Rec.**	**Prec.**	**Disc.**
CWE-23 Rel. Path T.	RQ2	11%	100%	11%	4%	100%	4%	86%	86%	72%
	RQ3	19%	100%	19%	47%	100%	47%	100%	88%	86%
CWE-36 Abs. Path T.	RQ2	11%	100%	11%	4%	100%	4%	86%	86%	72%
	RQ3	19%	100%	19%	40%	100%	40%	100%	88%	86%
Cross-Site Scripting										
		Rec.	**Prec.**	**Disc.**	**Rec.**	**Prec.**	**Disc.**	**Rec.**	**Prec.**	**Disc.**
CWE-80 Basic XSS	RQ2	11%	100%	11%	3%	100%	3%	100%	88%	86%
	RQ3	19%	100%	19%	46%	100%	46%	89%	100%	86%
CWE-81 XSS Error	RQ2	11%	100%	11%	6%	100%	6%	100%	88%	86%
	RQ3	19%	100%	19%	46%	100%	46%	89%	100%	86%
CWE-83 XSS Attrib.	RQ2	11%	100%	11%	6%	100%	6%	100%	88%	86%
	RQ3	19%	100%	19%	46%	100%	46%	89%	100%	86%

Summary

- Many Web app vulnerabilities are about details

- Developers need to understand the risks and to develop secure code from the first place

- Tools to help developers are not perfect and need improvements