Week 8 – Web Application Hacking

Web Application Security Testing Tools

Web Application Security Testing Tools

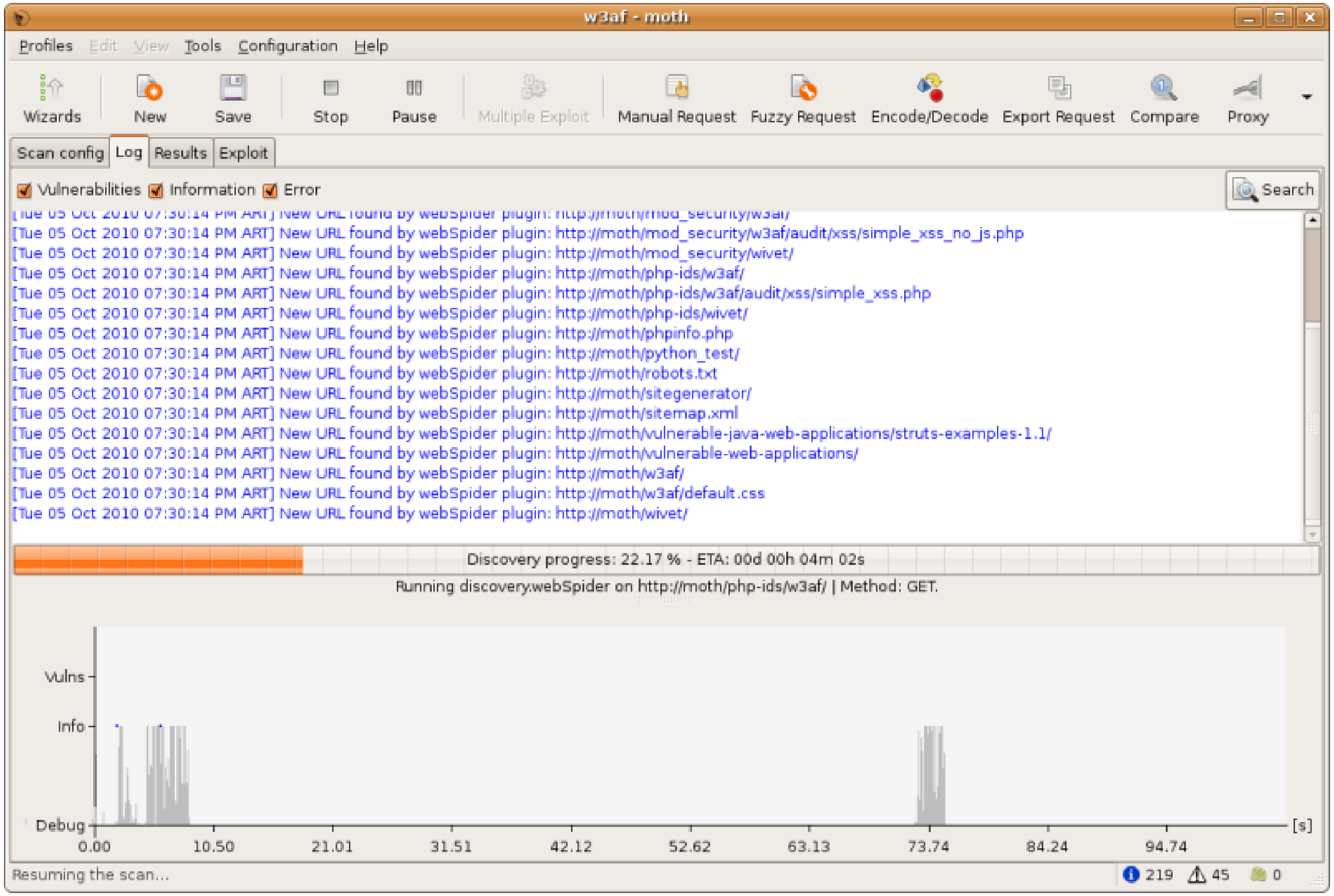
- Two types of security testing tools:
  - Active: send out requests and test for vulnerability
    - E.g. Web application security scanners
  - Passive: Intercept, manipulate or listen to web traffics, and detect for
    - E.g. Web proxies
- May need specific tools in different environment

- Caution!
  - No tools are 100% safe
  - Applications may not behave normally in security testing due to the their bugs, flaws or special feature / logical flow
  - Be extra careful when testing in production environment
    - Using staging / testing environment, if available, is good for both application owner and security tester

w3af (1)

- Automatic web application scanner
- In both CLI and GUI
- Open Source
- Website: http://w3af.sourceforge.net/

w3af (2)



```
[Tue 05 Oct 2010 07:30:14 PM ART] New URL found by webSpider plugin: http://moth/mod_security/w3af/
[Tue 05 Oct 2010 07:30:14 PM ART] New URL found by webSpider plugin: http://moth/mod_security/w3af/audit/xss/simple_xss_no_js.php
[Tue 05 Oct 2010 07:30:14 PM ART] New URL found by webSpider plugin: http://moth/mod_security/wivet/
[Tue 05 Oct 2010 07:30:14 PM ART] New URL found by webSpider plugin: http://moth/php-ids/w3af/
[Tue 05 Oct 2010 07:30:14 PM ART] New URL found by webSpider plugin: http://moth/php-ids/w3af/audit/xss/simple_xss.php
[Tue 05 Oct 2010 07:30:14 PM ART] New URL found by webSpider plugin: http://moth/php-ids/wivet/
[Tue 05 Oct 2010 07:30:14 PM ART] New URL found by webSpider plugin: http://moth/phpinfo.php
[Tue 05 Oct 2010 07:30:14 PM ART] New URL found by webSpider plugin: http://moth/python_test/
[Tue 05 Oct 2010 07:30:14 PM ART] New URL found by webSpider plugin: http://moth/robots.txt
[Tue 05 Oct 2010 07:30:14 PM ART] New URL found by webSpider plugin: http://moth/sitegenerator/
[Tue 05 Oct 2010 07:30:14 PM ART] New URL found by webSpider plugin: http://moth/sitemap.xml
[Tue 05 Oct 2010 07:30:14 PM ART] New URL found by webSpider plugin: http://moth/vulnerable-java-web-applications/struts-examples-1.1/
[Tue 05 Oct 2010 07:30:14 PM ART] New URL found by webSpider plugin: http://moth/vulnerable-web-applications/
[Tue 05 Oct 2010 07:30:14 PM ART] New URL found by webSpider plugin: http://moth/w3af/
[Tue 05 Oct 2010 07:30:14 PM ART] New URL found by webSpider plugin: http://moth/w3af/default.css
[Tue 05 Oct 2010 07:30:14 PM ART] New URL found by webSpider plugin: http://moth/wivet/
```

Discovery progress: 22.17 % - ETA: 00d 00h 04m 02s

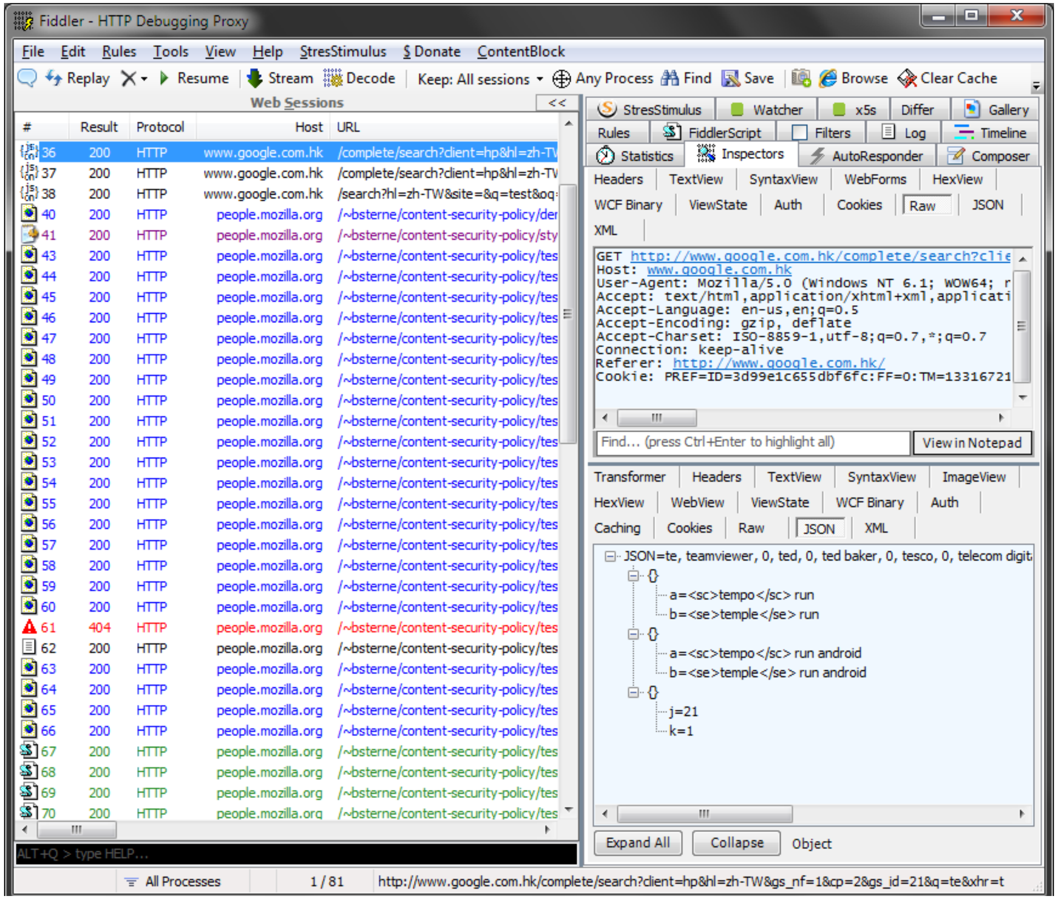
Running discovery.webSpider on http://moth/php-ids/w3af/ | Method: GET.

Resuming the scan...

Fiddler (1)

- “Web debugging proxy” – mainly a passive security testing tool
- Allows user to intercept and manipulate data
- Easily Extendable – you can write your own plugins for enhancement
- Website: http://www.fiddler2.com/fiddler2/

Fiddler (2)



#	Result	Protocol	Host	URL
36	200	HTTP	www.google.com.hk	/complete/search?client=hp&hl=zh-TW
37	200	HTTP	www.google.com.hk	/complete/search?client=hp&hl=zh-TW
38	200	HTTP	www.google.com.hk	/search?hl=zh-TW&site=&q=test&oq=
40	200	HTTP	people.mozilla.org	/~bsterne/content-security-policy/der
41	200	HTTP	people.mozilla.org	/~bsterne/content-security-policy/sty
43	200	HTTP	people.mozilla.org	/~bsterne/content-security-policy/tes
44	200	HTTP	people.mozilla.org	/~bsterne/content-security-policy/tes
45	200	HTTP	people.mozilla.org	/~bsterne/content-security-policy/tes
46	200	HTTP	people.mozilla.org	/~bsterne/content-security-policy/tes
47	200	HTTP	people.mozilla.org	/~bsterne/content-security-policy/tes
48	200	HTTP	people.mozilla.org	/~bsterne/content-security-policy/tes
49	200	HTTP	people.mozilla.org	/~bsterne/content-security-policy/tes
50	200	HTTP	people.mozilla.org	/~bsterne/content-security-policy/tes
51	200	HTTP	people.mozilla.org	/~bsterne/content-security-policy/tes
52	200	HTTP	people.mozilla.org	/~bsterne/content-security-policy/tes
53	200	HTTP	people.mozilla.org	/~bsterne/content-security-policy/tes
54	200	HTTP	people.mozilla.org	/~bsterne/content-security-policy/tes
55	200	HTTP	people.mozilla.org	/~bsterne/content-security-policy/tes
56	200	HTTP	people.mozilla.org	/~bsterne/content-security-policy/tes
57	200	HTTP	people.mozilla.org	/~bsterne/content-security-policy/tes
58	200	HTTP	people.mozilla.org	/~bsterne/content-security-policy/tes
59	200	HTTP	people.mozilla.org	/~bsterne/content-security-policy/tes
60	200	HTTP	people.mozilla.org	/~bsterne/content-security-policy/tes
61	404	HTTP	people.mozilla.org	/~bsterne/content-security-policy/tes
62	200	HTTP	people.mozilla.org	/~bsterne/content-security-policy/tes
63	200	HTTP	people.mozilla.org	/~bsterne/content-security-policy/tes
64	200	HTTP	people.mozilla.org	/~bsterne/content-security-policy/tes
65	200	HTTP	people.mozilla.org	/~bsterne/content-security-policy/tes
66	200	HTTP	people.mozilla.org	/~bsterne/content-security-policy/tes
67	200	HTTP	people.mozilla.org	/~bsterne/content-security-policy/tes
68	200	HTTP	people.mozilla.org	/~bsterne/content-security-policy/tes
69	200	HTTP	people.mozilla.org	/~bsterne/content-security-policy/tes
70	200	HTTP	people.mozilla.org	/~bsterne/content-security-policy/tes

```
GET http://www.google.com.hk/complete/search?clie
Host: www.google.com.hk
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; r
Accept: text/html,application/xhtml+xml,applicati
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip, deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Connection: keep-alive
Referer: http://www.google.com.hk/
Cookie: PREF=ID=3d99e1c655dbf6fc:FF=0:TM=13316721
```

```
JSON=te, teamviewer, 0, ted, 0, ted baker, 0, tesco, 0, telecom digit
  {}
    a=<sc>tempo</sc> run
    b=<se>temple</se> run
  {}
    a=<sc>tempo</sc> run android
    b=<se>temple</se> run android
  {}
    j=21
    k=1
```

http://www.google.com.hk/complete/search?client=hp&hl=zh-TW&gs_nf=1&cp=2&gs_id=21&q=te&xhr=t

Other web debugging proxies

- Paros
- ZAP
- Burp suit
- Ratproxy
- OWASP WebScarab
- etc

Features are quite similar: feel free to use your favorite one

Browser plugins

- Manipulates the DOM or traffic in browser
- Easy to use due to the integration of browsing experience
- Both active and passive tools are available
- Example in Firefox
  - FireBugs
  - HackBar
  - WebDeveloper Toolbar
  - etc
- Using the “Developer Tools” in modern browsers is also a good idea
- Alternatives also available in other common browsers

IMPORTANT NOTES

BE ETHICAL!!!

Do’s and Don’ts

- Do’s
  - DO get written permission from the owner before performing any security tests
  - DO stop at mutually agreed point of intrusion, if you can
  - DO keep the vulnerability or other sensitive information carefully
  - DO report the information to the owner if you find some vulnerability “by chance”
  - DO know your liability and rights

- Don’ts
  - DON’T publish the vulnerability information without permission of the owner
  - DON’T blackmail the owner with what you have
  - DON’T think you won’t be traced and caught

You are warned! ☺

Lab

WARM UP EXERCISE

OWASP Top 10

OWASP Top 10 Web Application Security Risk (2007 version)

Open Web Application Security Projects (OWASP) defines Top 10 Web Application Security Risks (2007 version):

- A1: Cross Site Scripting (XSS)
- A2: Injection Flaws
- A3: Malicious File Execution
- A4: Insecure Direct Object Reference
- A5: Cross Site Request Forgery (CSRF)
- A6: Information Leakage and Improper Error Handling
- A7: Broken Authentication and Session Management
- A8: Insecure Cryptographic Storage
- A9: Insecure Communication
- A10: Failure to Restrict URL Access

Source: https://www.owasp.org/index.php/Top_10_2007

OWASP
The Open Web Application Security Project

OWASP Top 10 Web Application Security Risk (2010 version)

Top 10 Web Application Security Risk 2010 version:

- A1: Injection
- A2: Cross-site Scripting (XSS)
- A3: Broken Authentication and Session Management
- A4: Insecure Direct Object References
- A5: Cross-site Request Forgery (CSRF)
- A6: Secure Misconfiguration
- A7: Insecure Cryptographic Storage
- A8: Failure to Restrict URL Access
- A9: Insufficient Transport Layer Protection
- A10: Unvalidated Redirects and Forwards

Source: https://www.owasp.org/index.php/Top_10_2010

OWASP
The Open Web Application Security Project

OWASP Top 10 Web Application Security Risk (2013 version)

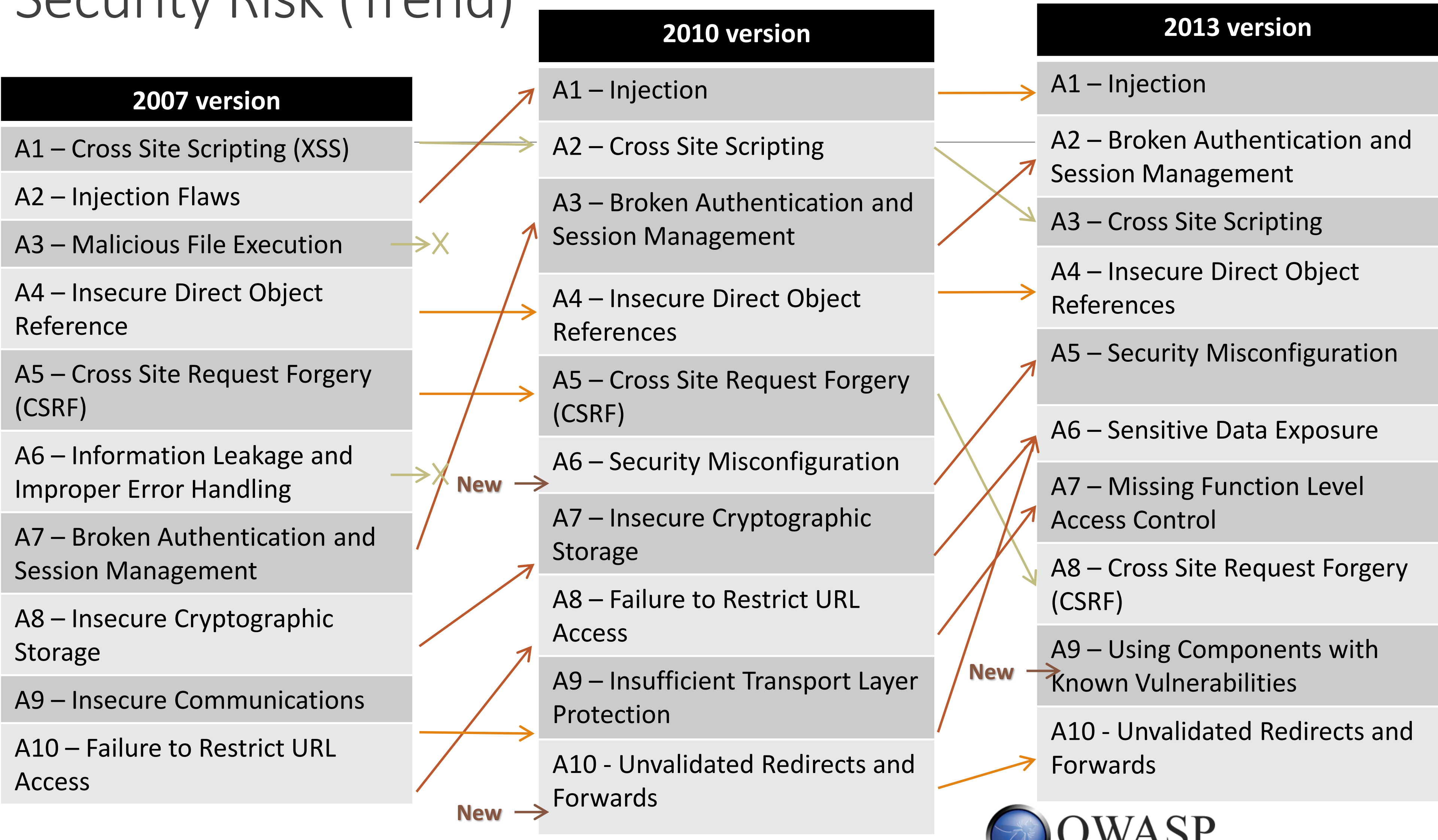
Top 10 Web Application Security Risk 2013 release candidate version:

- A1: Injection
- A2: Broken Authentication and Session Management
- A3: Cross-site Scripting (XSS)
- A4: Insecure Direct Object References
- A5: Secure Misconfiguration
- A6: Sensitive Data Exposure
- A7: Missing Function Level Access Control
- A8: Cross-site Request Forgery (CSRF)
- A9: Using Components with Known Vulnerabilities
- A10: Unvalidated Redirects and Forwards

Source: https://www.owasp.org/index.php/Top_10_2013

OWASP
The Open Web Application Security Project

OWASP Top 10 Web Application Security Risk (Trend)



2007 version	2010 version	2013 version
A1 – Cross Site Scripting (XSS)	A1 – Injection	A1 – Injection
A2 – Injection Flaws	A2 – Cross Site Scripting	A2 – Broken Authentication and Session Management
A3 – Malicious File Execution	A3 – Broken Authentication and Session Management	A3 – Cross Site Scripting
A4 – Insecure Direct Object Reference	A4 – Insecure Direct Object References	A4 – Insecure Direct Object References
A5 – Cross Site Request Forgery (CSRF)	A5 – Cross Site Request Forgery (CSRF)	A5 – Security Misconfiguration
A6 – Information Leakage and Improper Error Handling	A6 – Security Misconfiguration	A6 – Sensitive Data Exposure
A7 – Broken Authentication and Session Management	A7 – Insecure Cryptographic Storage	A7 – Missing Function Level Access Control
A8 – Insecure Cryptographic Storage	A8 – Failure to Restrict URL Access	A8 – Cross Site Request Forgery (CSRF)
A9 – Insecure Communications	A9 – Insufficient Transport Layer Protection	A9 – Using Components with Known Vulnerabilities
A10 – Failure to Restrict URL Access	A10 - Unvalidated Redirects and Forwards	A10 - Unvalidated Redirects and Forwards

OWASP
The Open Web Application Security Project

OWASP Top 10 & Countermeasures

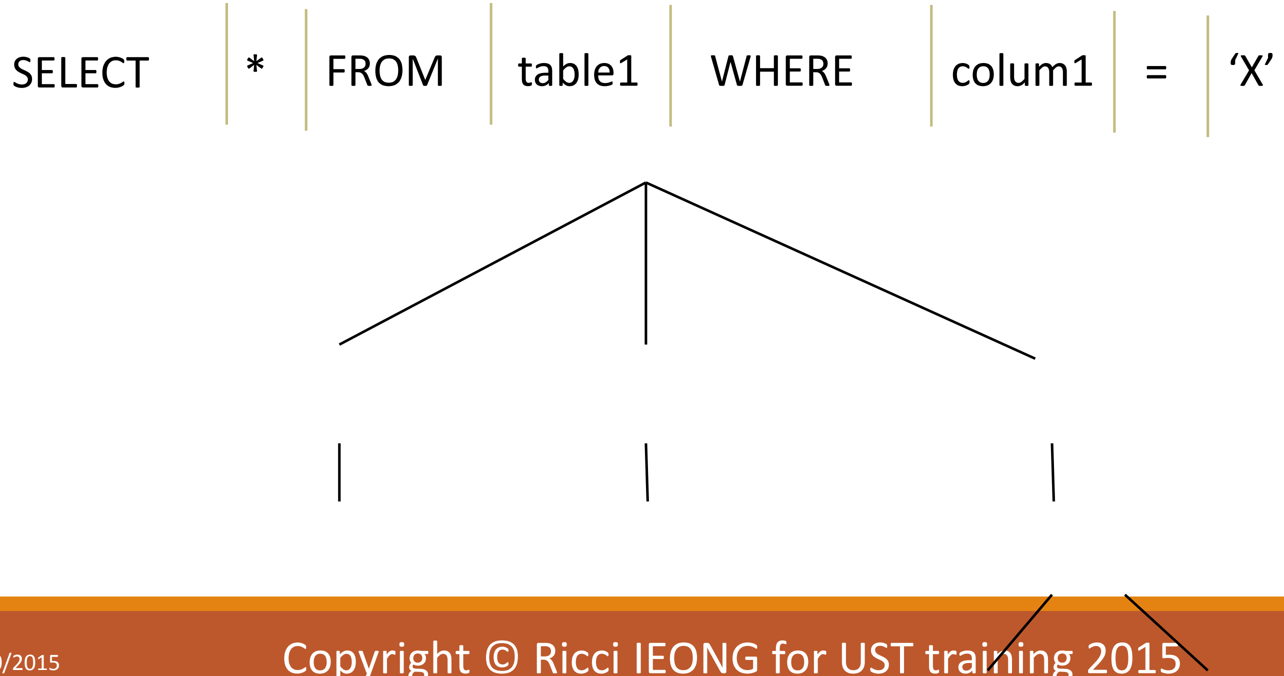
A1 – INJECTION

Parsing

- The processing of converting text to data structure representing the object

1. Identify keywords in the string
2. Split the text strings into tokens and then arrange to the data structure (e.g. tree)

- Example:

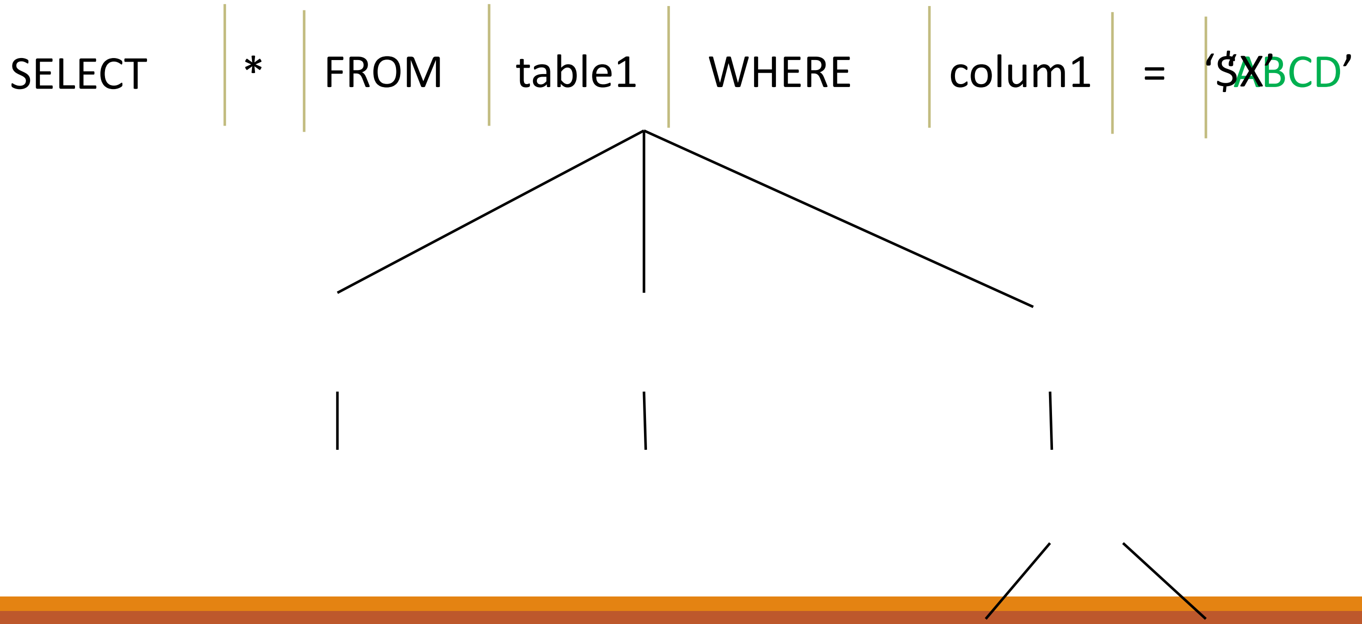


SELECT
*
FROM
table1
WHERE
colum1
=
‘X’

Parsing with variables

- If the target string (e.g. a SQL statement) contains variable…
  1. **Evaluate the variables**
  2. Identify keywords in the string
  3. Split the text strings into tokens and then arrange to the data structure (e.g. tree)

$X = ABCD



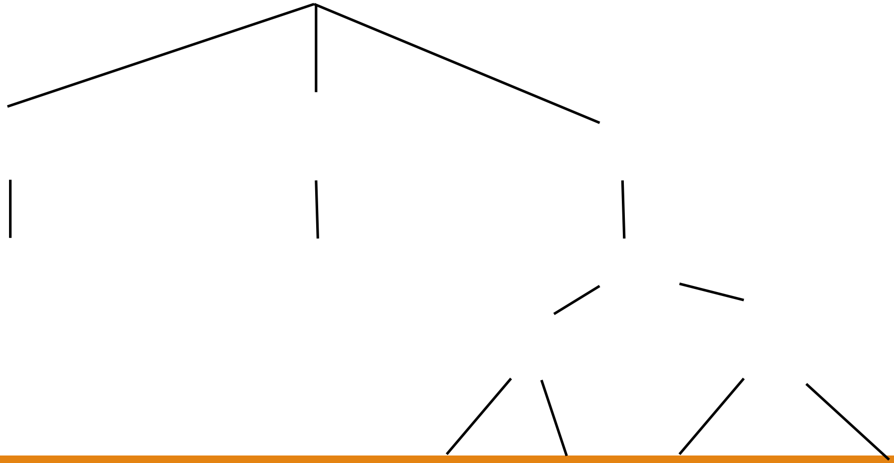
SELECT | * | FROM | table1 | WHERE | colum1 | = | ‘$X’ ABCD’

Problem…

- What if the string contain special “keywords”…?
  1. Evaluate the variables
  2. Identify keywords in the string
  3. Split the text strings into tokens and then arrange to the data structure (e.g. tree)

$X = ABCD’ OR ‘1’=‘1

SELECT	*	FROM	table1	WHERE	colum1	=	‘$X’ ABCD’	OR	‘1’	=	‘1’



Injection Flaw

- Occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing unauthorized data
- ***Definition:*** Change of original semantic structure of by injecting special characters to cheat the string parser
- Most common injections: SQL Injection

SQL Injection

- Example
  - SELECT * FROM user_table WHERE username=‘$username’ AND password=‘$password’;
  - case (1): $usernmae → userA, $password → abc123:
    - SELECT * FROM user_table WHERE username=‘userA’ AND password=‘abc123’;
  - case (2): $usernmae → userA, $password → a’ OR ‘’=‘:
    - SELECT * FROM user_table WHERE username=‘userA’ AND password=‘a’ OR ‘’=‘’;
    - Injection!

SQL Injection

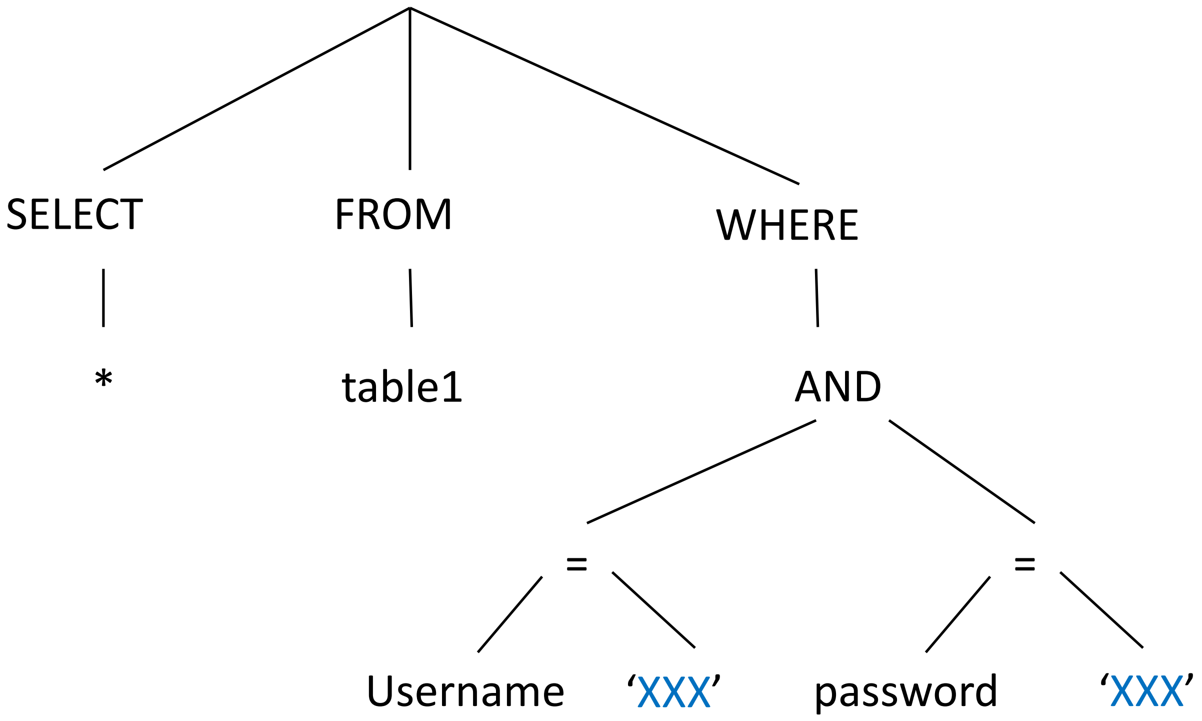
```
SELECT *
FROM user_table
WHERE
        Username=‘userA’
        AND password=‘a’
        OR ‘’=‘’;
```

$username → userA

$password → a’ OR ‘’=‘

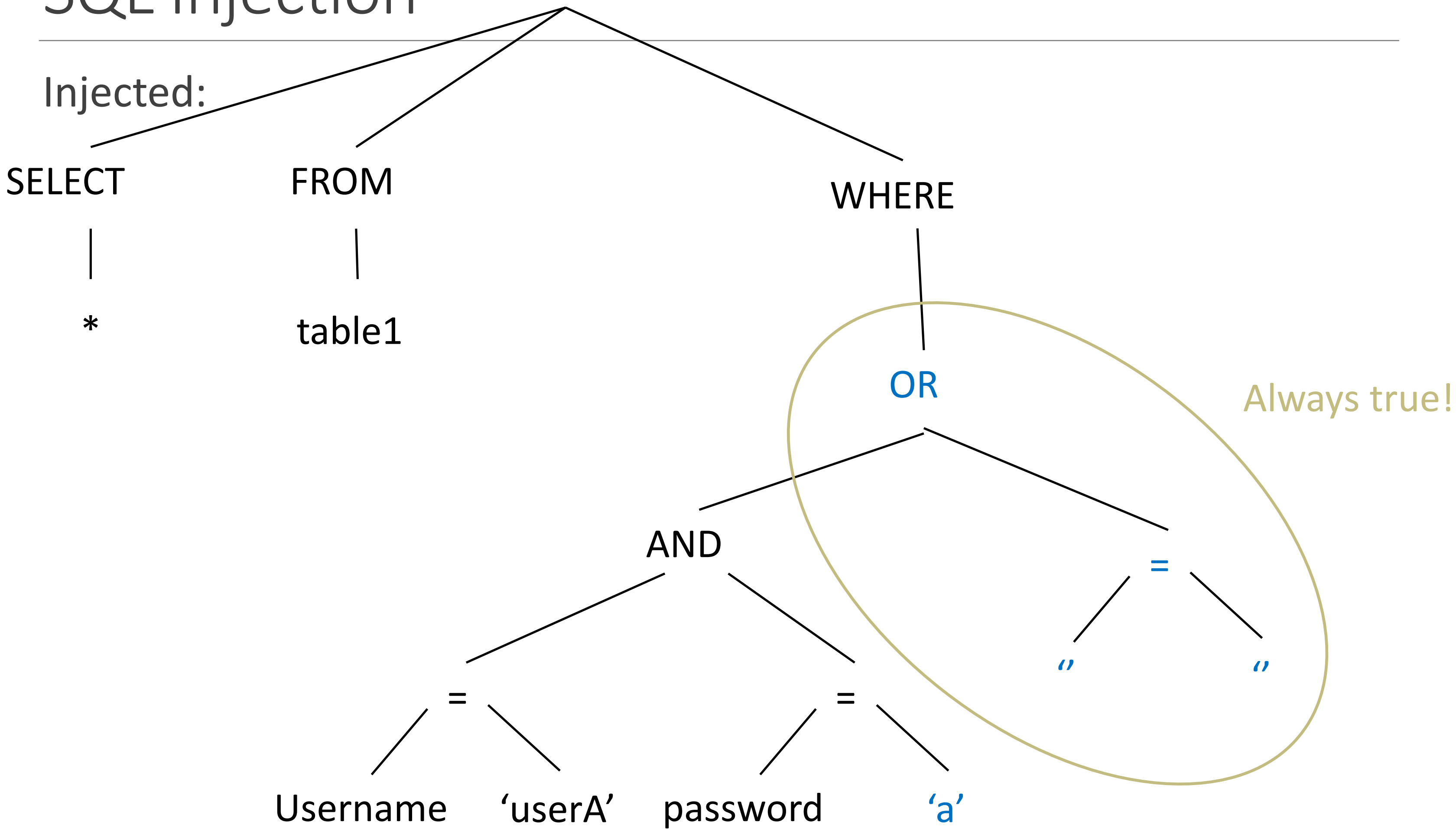
SQL Injection

Original:



SQL Injection

Injected:



SELECT

*

FROM

table1

WHERE

OR

AND

=

Username

‘userA’

=

password

‘a’

=

“

“

Always true!

Stacked SQL statements

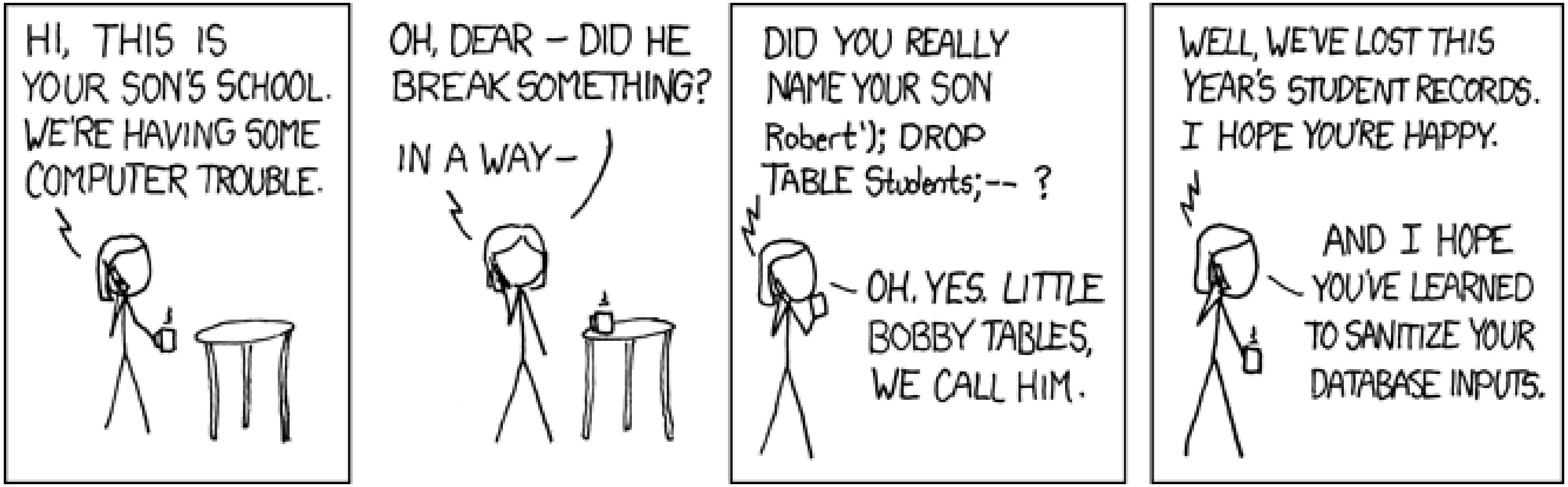
- Some SQL engine support “stacked” statements when running a query.
  - E.g.

    SELECT * FROM table1; INSERT INTO table1 (column1) VALUES (‘1’);

    is equivalent to running two queries separately

    SELECT * FROM table1;
    INSERT INTO table1 (column1) VALUES (‘1’);

  - The impact of SQL injection can be even more harmful


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.

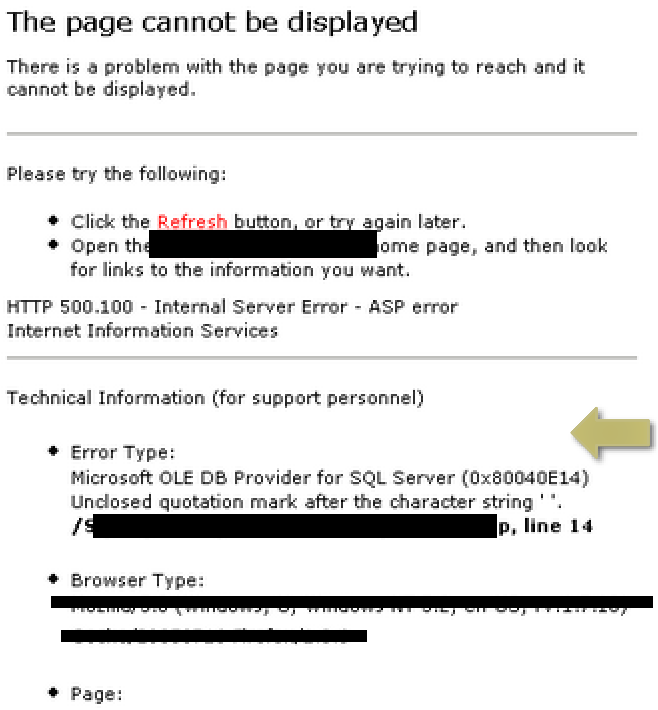
Source: http://xkcd.com/327/

SQL Injection 101

- However, external parties (attacker and black box pen tester) may not know the exact SQL statement used in the the application
- Question: How to find out a “correct” injection to the SQL statement?
- Possibilities:

1. Trial and error!
2. Testing some “magic” strings that usually works
   - ‘ or ‘’=‘’; --
   - or 1=1;--
   - etc
3. By observing error messages returned by the applications

Error message is the friend of attacker!



The page cannot be displayed

There is a problem with the page you are trying to reach and it cannot be displayed.

Please try the following:

- Click the Refresh button, or try again later.
- Open the [redacted]ome page, and then look for links to the information you want.

HTTP 500.100 - Internal Server Error - ASP error
Internet Information Services

Technical Information (for support personnel)

- Error Type:
  Microsoft OLE DB Provider for SQL Server (0x80040E14)
  Unclosed quotation mark after the character string ''.
  /S[redacted]p, line 14
- Browser Type:
  [redacted]
- Page:

How to prevent?

1. Hide the error message!
   - Yes it may make the attack harder to perform → “delay” control
   - However, it is not impossible → *blind SQL injection*

2. Sanitizing user provided content
   - Filter out / escape special characters like single quotes, and etc, *at server side*
     - You need to know the exact set of characters to be filtered out ☺
     - Whitelist approach is always better than blacklist approach

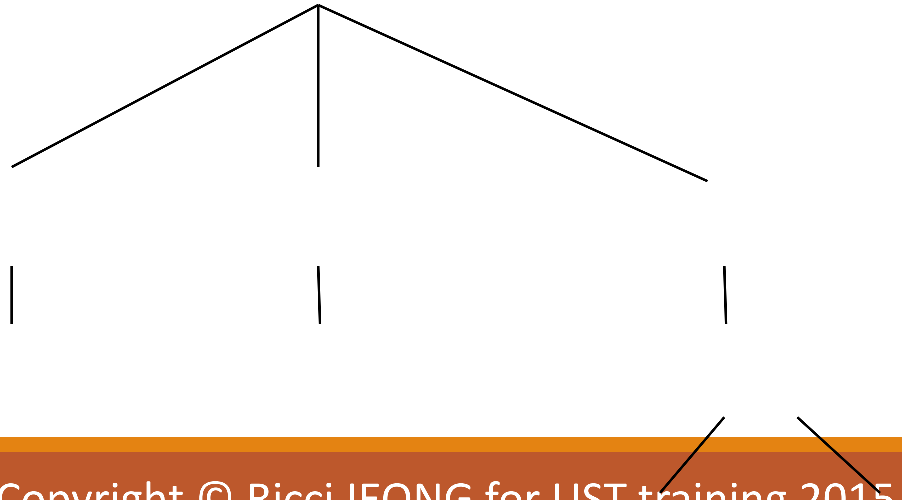
3. Parameterized SQL statement
   - Pre-build the SQL statement semantic structure before evaluating the variables

Parsing with parameterized variables

- If the target string (e.g. a SQL statement) contains parameterized variable
  1. Identify keywords in the string
  2. Split the text strings into tokens and then arrange to the data structure (e.g. tree)
  3. **Evaluate the variables**

$X = ABCD' OR '1'='1

SELECT	*	FROM	table1	WHERE	colum1	=	'$X'



More SQL Injection

- What if error messages are masked / removed?

1. Possibility #1: See if any place can be used to extract values:
   - INSERT INTO table1 (c1, c2), values (‘test’, (SELECT c3 FROM table2))
2. Possibility #2: Observe the HTTP status code
   - By default return a HTTP 500 when SQL error is occurred
   - If this is
3. Possibility #3: Observe other application specific properties…
   - Response time?
     - Successful queries usually take longer time to complete than failed queries
     - The difference is more observable if the returned data size is large enough
   - Some special actions performed by the application?
     - Redirect to certain page, e.g. login page?

#2 & #3 → Blind SQL injection

What can we do with SQL injection?

1. Retrieve information from database…
   - SELECT c1 FROM table1 WHERE id=‘1’ UNION SELECT c2 FROM table2;--’
2. Retrieve system information
   - SELECT c1 FROM table1 WHERE id=‘1’ UNION SELECT table_name FROM information_schema.tables;--’
3. Modify database
   - SELECT c1 FROM table1 WHERE id=‘1’; UPDATE table1 SET c1=‘a’;
4. Execute commands, if the database supports…
   - SELECT c1 FROM table1 WHERE id=‘1’; EXEC xp_cmdshell ‘net user’;--’
5. Many more…
   - Cheat Sheets: http://pentestmonkey.net/category/cheat-sheet/sql-injection

Use your creativity!!! ☺

Lab

BASIC SQL INJECTION AND COUNTERMEASURES

Only SQL Injection?

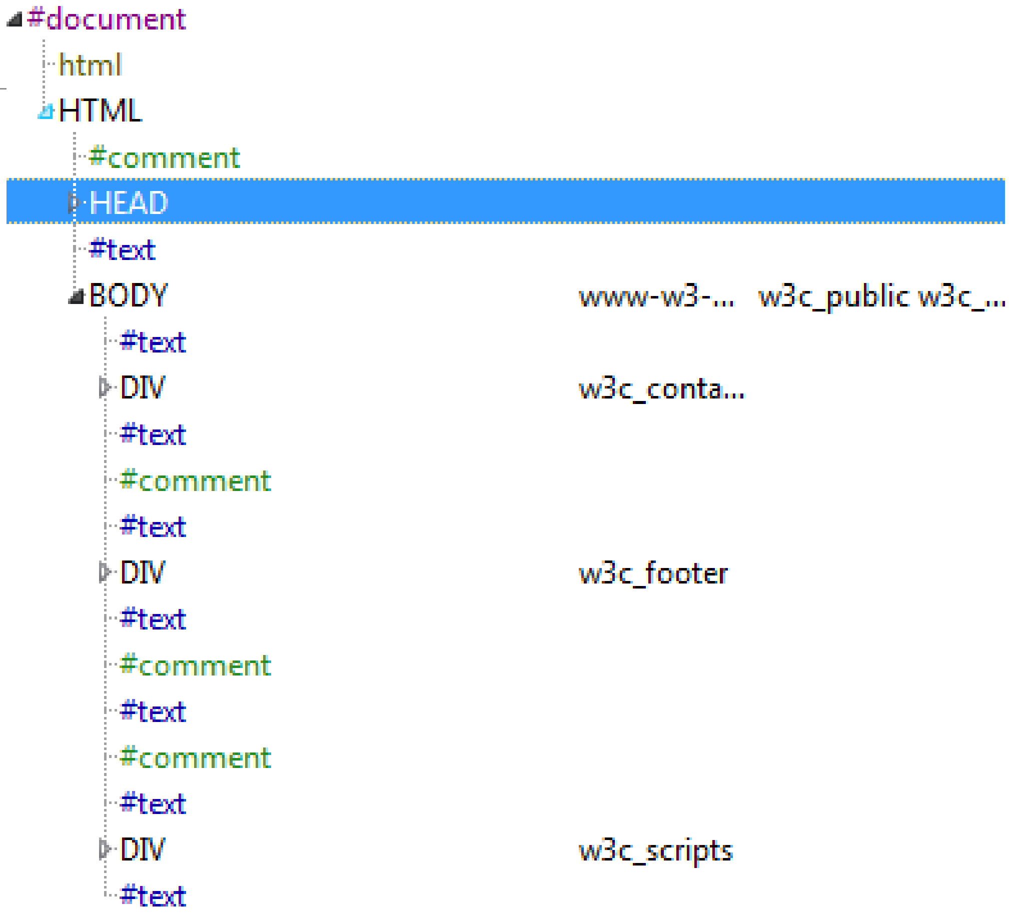
- **NO!**
- ***Definition:*** Change of original semantic structure of by injecting special characters to cheat the string parser
- Apply to all kinds of parsing:
  - E.g. XML Injection, LDAP Injection, etc

OWASP Top 10 & Countermeasures

A3 – CROSS-SITE SCRIPTING

HTML & DOM

- HTML is *parsed* by rendering engine in browsers into a DOM tree structure
- Can we use injection to change the DOM structure of the page?
- YES!



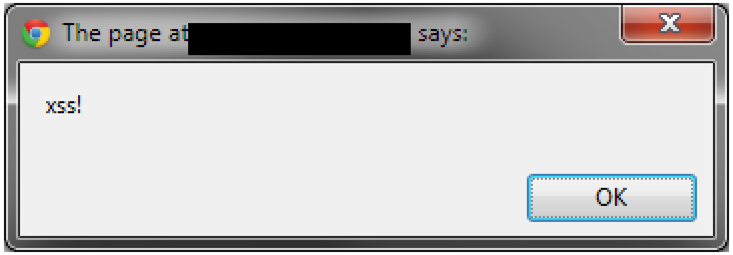
```
#document
  html
  HTML
    #comment
    HEAD
    #text
    BODY                                   www-w3-...  w3c_public w3c_...
      #text
      DIV                                  w3c_conta...
      #text
      #comment
      #text
      DIV                                  w3c_footer
      #text
      #comment
      #text
      #comment
      #text
      DIV                                  w3c_scripts
      #text
```

Cross Site Scripting (XSS)

- Occur whenever an application takes untrusted data and sends it to a web browser without proper validation and escaping

- Allows attackers to execute scripts in the victim’s browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites

- Exploit the *trust of user’s browser* from the data returned by the server



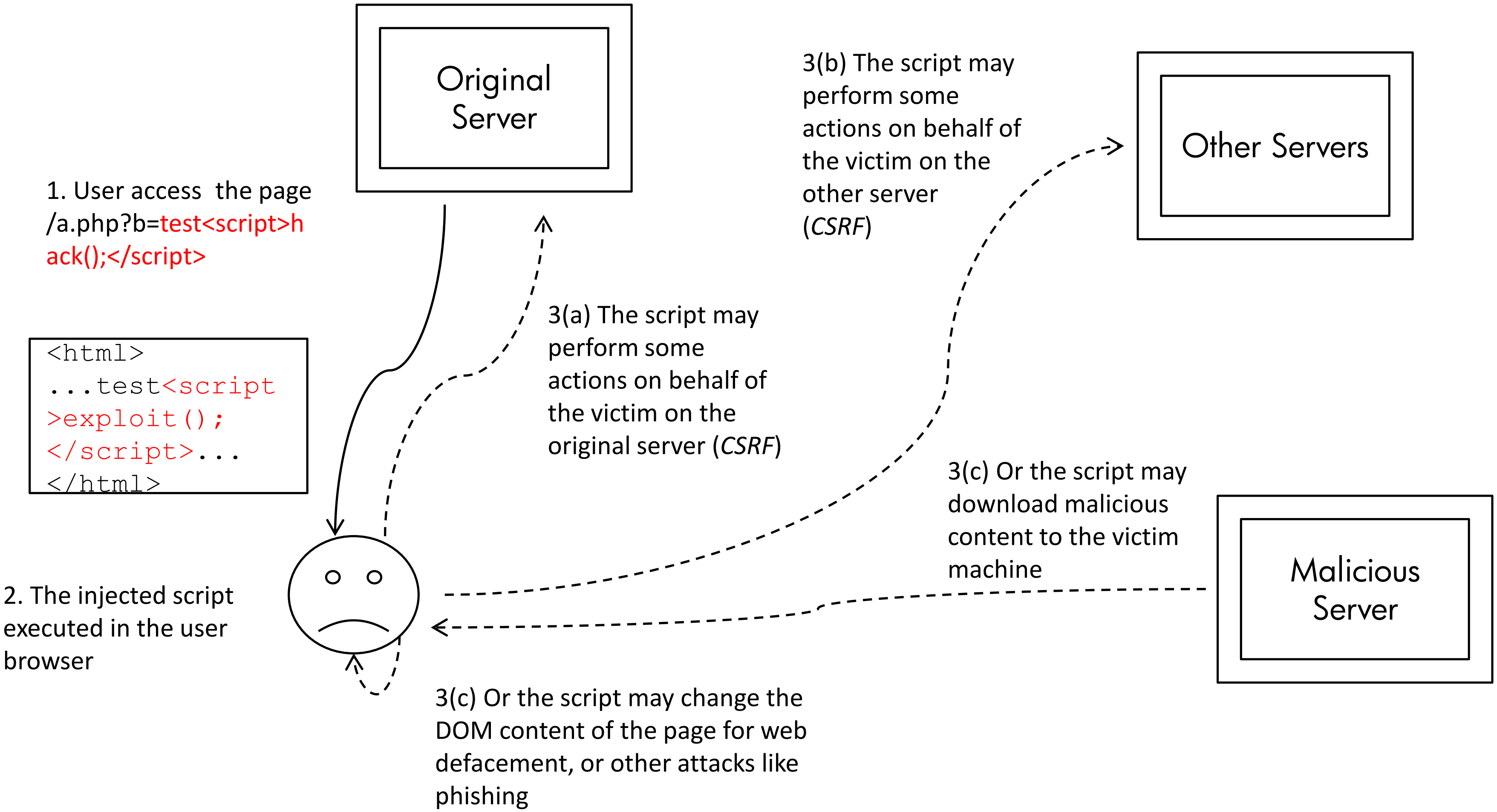
The page at [redacted] says:

xss!

Type of XSS

- 3 types of XSS
  - Type 0: DOM-based XSS
    - by directly modifying the DOM structure, e.g.
      - Cheating the victim to click on the links javascript:alert(‘xss!’)
  - Type 1: Reflected XSS
    - The injected script is at non-persistent value
      - E.g. URL parameters
  - Type 2: Stored XSS / Persistence XSS
    - The injected script is stored persistently
      - E.g. database, file, browser cookies

XSS Illustration



Original Server

Other Servers

Malicious Server

1. User access the page /a.php?b=test<script>hack();</script>

```
<html>
...test<script
>exploit();
</script>...
</html>
```

2. The injected script executed in the user browser

3(a) The script may perform some actions on behalf of the victim on the original server (*CSRF*)

3(b) The script may perform some actions on behalf of the victim on the other server (*CSRF*)

3(c) Or the script may download malicious content to the victim machine

3(c) Or the script may change the DOM content of the page for web defacement, or other attacks like phishing

Common XSS techniques

- Directly inject browser content script:
  - `<script> var x='test';exploit();var x='test'</script>`

- Inject HTML node
  - `<input type='text' value='test'><script>exploit()</script><img src=''>`

- Inject HTML node (trigger by DOM event)
  - `<input type='text' value='test'><img src='notexist' onerror='javascript:exploit()'>`

- Which technique to use depends on situations…

XSS Challenges

- Sound easy, but really depends on situations…
  - When the script should be executed?
    - order of execution matters (a lot!)
  - How to avoid script error after injection?
    - The DOM structure or the script syntax may be changed
    - The script may be injected at multiple place at the same time

- No golden rule!

- Be creative! ☺

- Practice makes perfect! ☺☺☺

How to prevent?

1. Golden rule: sanitizing all user provided content
   - Filter out / escape special characters like single quotes, and etc
     - You need to know the exact set of characters to be filtered out ☺
     - Whitelist approach is always better than blacklist approach
   - Verify the type of the value is a good idea

2. Minimize the impact of XSS
   - Do not store sensitive data in client side
   - Limit the access of user script to session cookies (setting the `HttpOnly` flag)
   - Disable TRACE/TRACK HTTP method that can bypass the `HttpOnly` restriction
   - Properly arrange the domain - cookies are restricted to domain only. Separate sensitive & non-sensitive service into two domain
   - Set proper Cross-domain Security Policies

OWASP Top 10 & Countermeasures

A8 – CROSS SITE REQUEST FORGERY

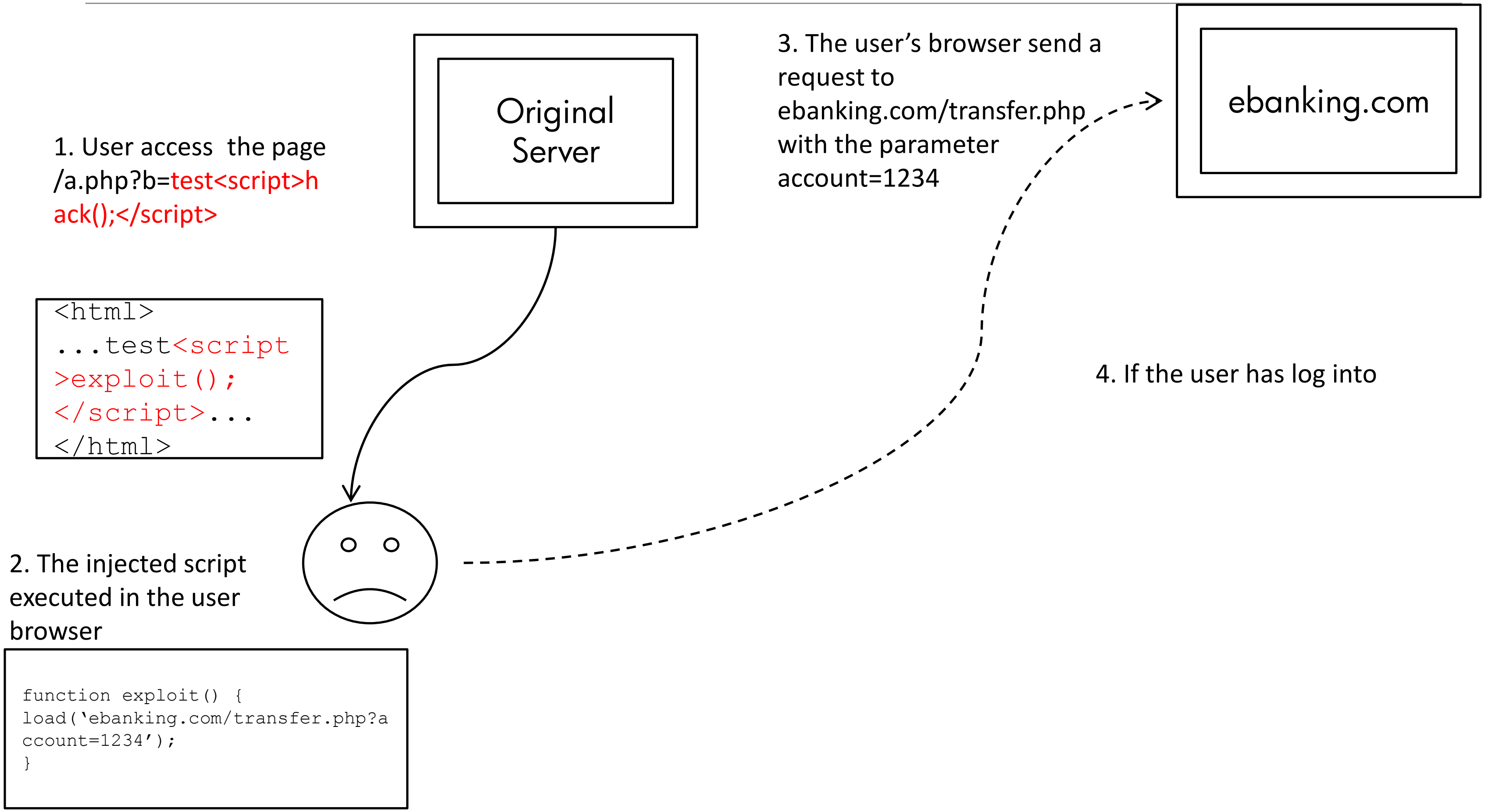
Cross-Site Request Forgery (CSRF)

- A CSRF attack forces a logged-on victim’s browser to send a forged HTTP request, including the victim’s session cookie and any other automatically included authentication information, to a vulnerable web application. This allows the attacker to force the victim’s browser to generate requests the vulnerable application thinks are legitimate requests from the victim

- Exploit the *trust of server* on the client

- Usually CSRF is achieved via other vulnerabilities like, XSS

CSRF Illustration



1. User access the page /a.php?b=test<script>hack();</script>

```
<html>
...test<script
>exploit();
</script>...
</html>
```

2. The injected script executed in the user browser

```
function exploit() {
load(‘ebanking.com/transfer.php?a
ccount=1234’);
}
```

3. The user’s browser send a request to ebanking.com/transfer.php with the parameter account=1234

4. If the user has log into

How to prevent?

1. Check the source of the requests on critical functions
   - `HTTP REFER header`
   - `HTTP ORIGIN header`

2. Using HTTP POST to submit data add *little bit difficulties* in exploiting CSRF than using HTTP GET to submit data

3. Using “CSRF-Tokens”

Lab

XSS/CSRF ATTACK AND COUNTERMEASURES

Reference Books

Related Content	Book	Chapter
W8, 9: Web Security	Guide to Computer Network Security (2015)	Chapter 6: Scripting and Security in Computer Networks
W8: E-business attack scenario, W8: web attack	Computer Security Handbook (2014)	Chapter 21: Web-based Vulnerabilities
Web Security	OWASP web site	OWASP Top 10
Injection, XSS attacks	Hacking Web Applications Exposed 3	Chapter 6 Input Injection Attacks