Web Application Vulnerabilities and In-secure Software Root Causes: The OWASP Top 10

Cincinnati Chapter Meeting
February 26th, 2008
Marco.Morana@OWASP.ORG

OWASP

The OWASP Foundation
http://www.owasp.org

Agenda

1. Application Security and The Medical Metaphor
2. Software Security From a Process Perspective
3. Software Security Strategy
4. Essential Elements For Secure Coding Standards/Guidelines
5. OWASP Top Ten 2007
6. Security Issues, Threats, Software Root Causes, Validations and Recommendations

Application Security And The Medical Metaphor

- Three dimensions of the application security problem from the perspective of a sick patient being visited by a doctor:
  - Symptoms
  - Risk factors
  - Root causes

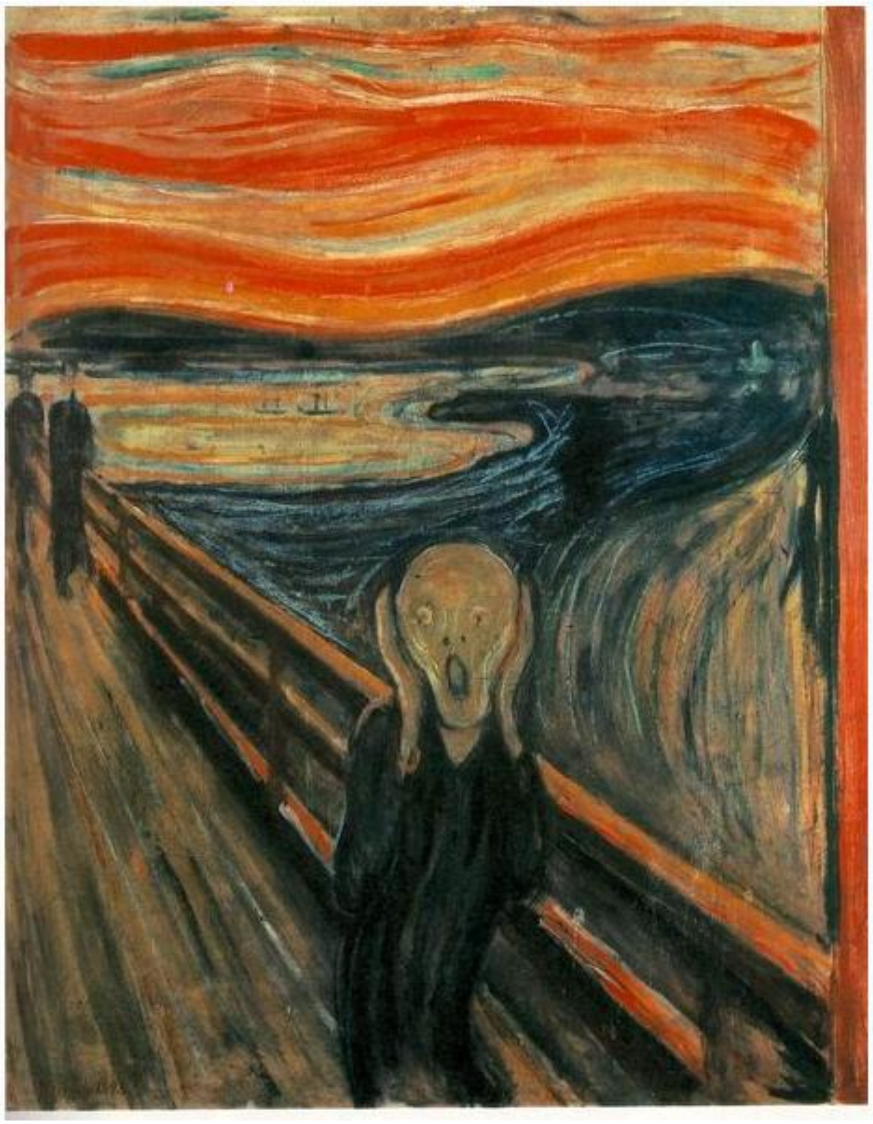
Symptoms, Causes and Risk Factors

- **The symptoms** are the insecure observed behavior of the application against potential vulnerabilities and exploits
- **The root causes** are security design flaws, security bugs (coding errors), insecure-configuration
- **The risk factors** are the quantifiable risks such as how much damage can be done, how easy is to reproduce the exploits, how many users are exposed and how easy is to discover the vulnerabilities

Symptoms Of Bad Application Security

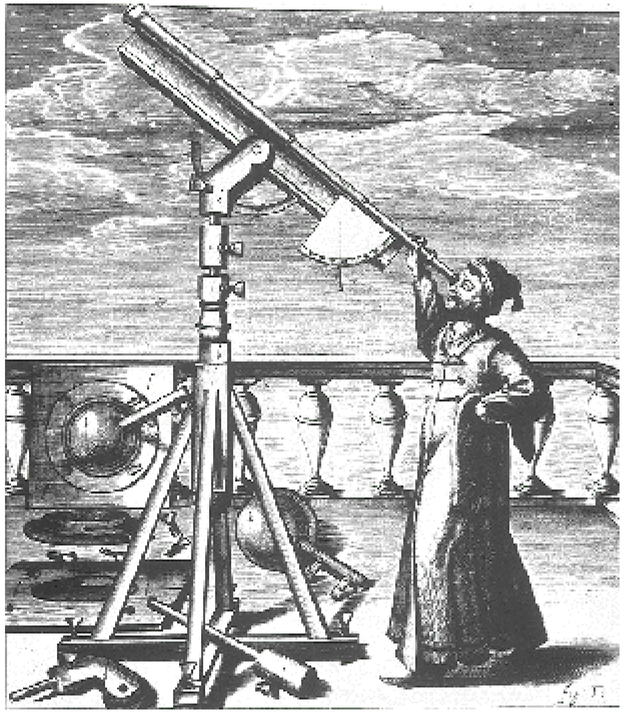
- A vulnerability in my ecommerce site allows a user to manipulate price values so a product get purchased for a lower price
- Some of my customers credit card numbers have been stolen and a financial fraud occurred because of weak encryption (WEP) between POS (Point Of Sale) terminal and branch servers
- TAXID, PII and account numbers of my customers got stolen by hackers exploiting a XSS of my web site via phishing attacks

How NO… I am Diagnosed a Cancer!

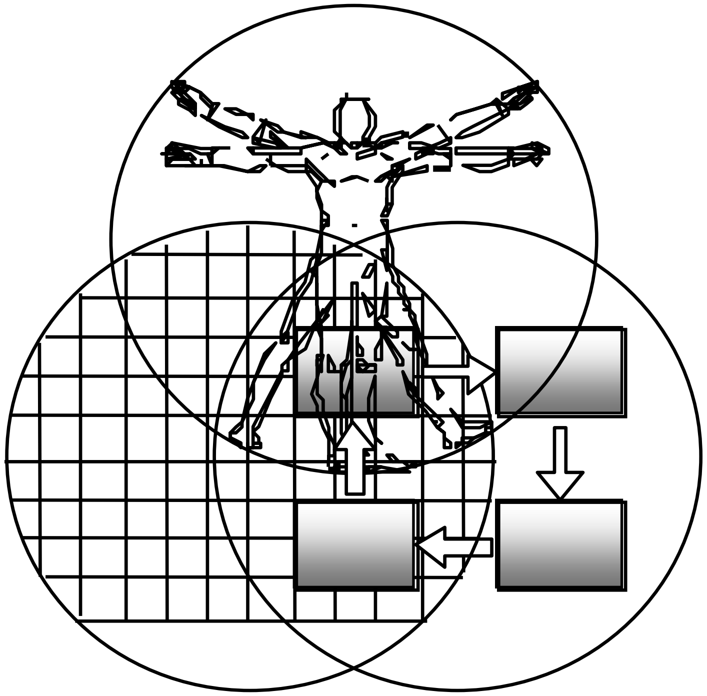


Do Not Panic, There is Cure!

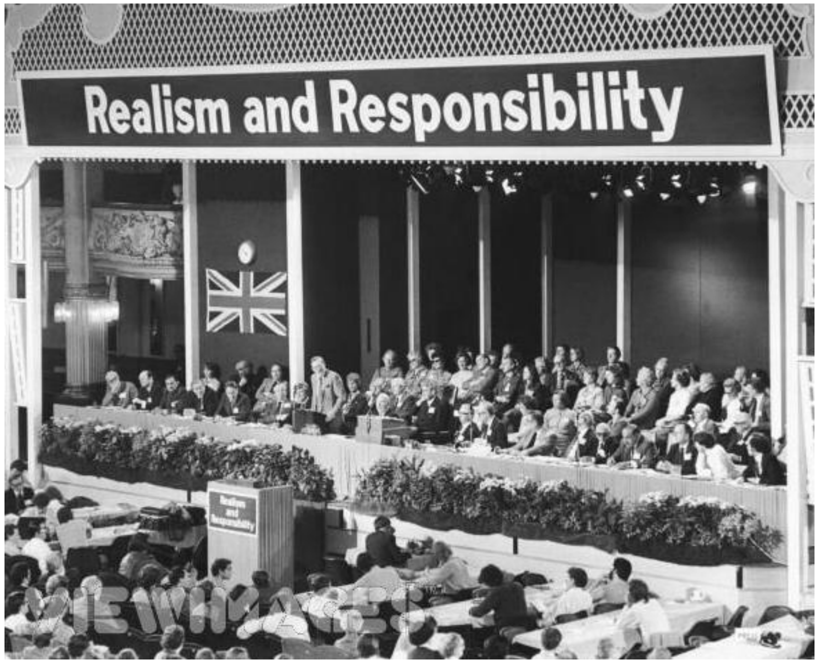
Focus on the Root Cause: Insecure software..



… but keep a 360 degree perspective: People, Process and Tools

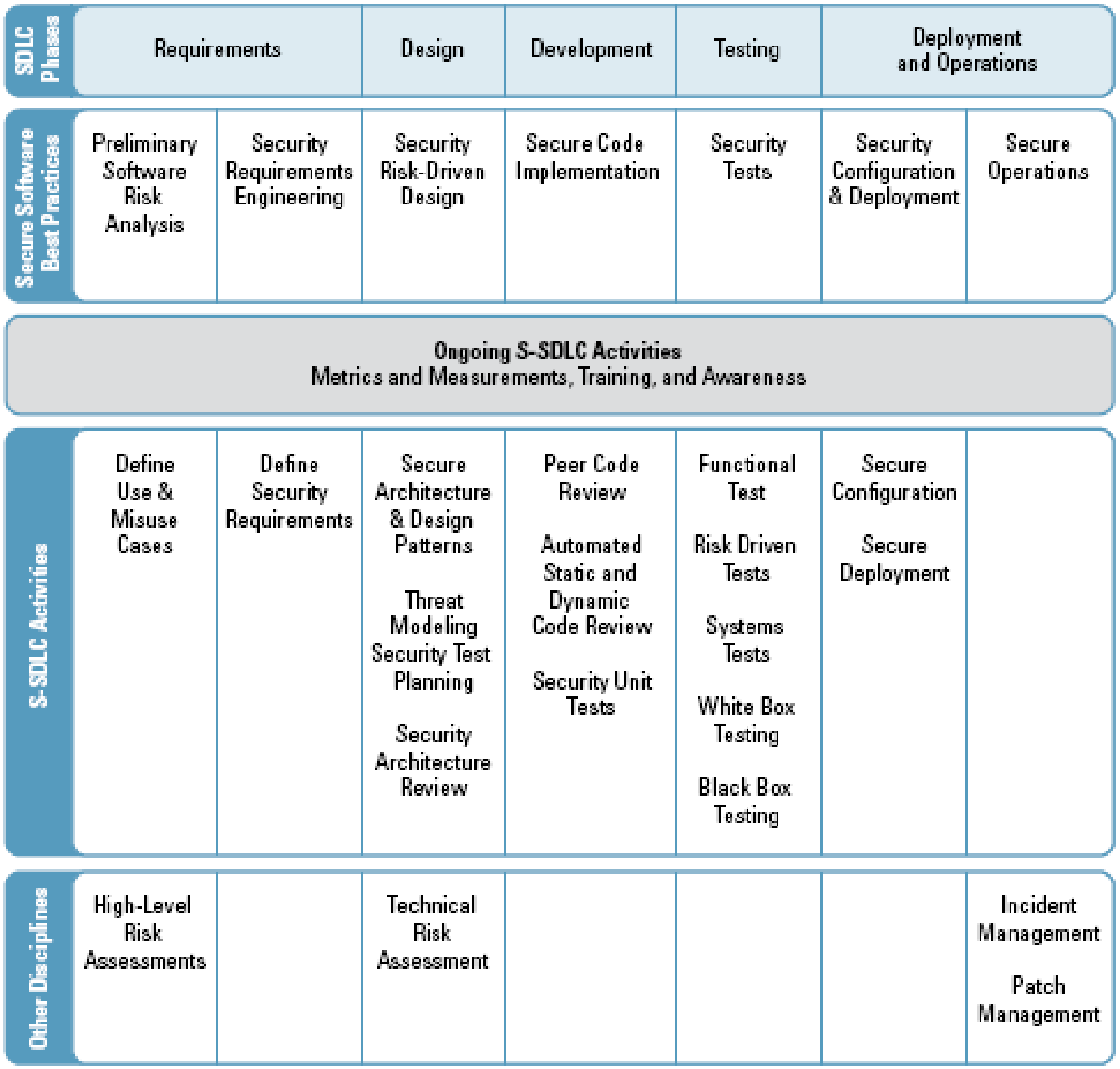


Software Security Strategy



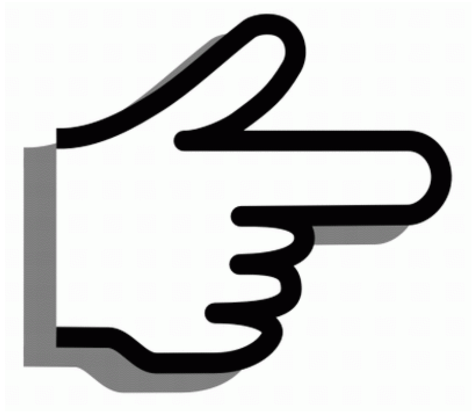
"If your software security practices are not yet mature be pragmatic and start making secure coding a responsibility for who builds software in your organization

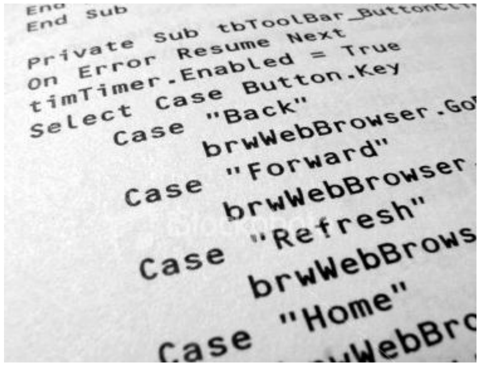
360 Deg Perspective: Security in the SDLC



SDLC Phases	Requirements		Design	Development	Testing	Deployment and Operations	
Secure Software Best Practices	Preliminary Software Risk Analysis	Security Requirements Engineering	Security Risk-Driven Design	Secure Code Implementation	Security Tests	Security Configuration & Deployment	Secure Operations
Ongoing S-SDLC Activities Metrics and Measurements, Training, and Awareness							
S-SDLC Activities	Define Use & Misuse Cases	Define Security Requirements	Secure Architecture & Design Patterns Threat Modeling Security Test Planning Security Architecture Review	Peer Code Review Automated Static and Dynamic Code Review Security Unit Tests	Functional Test Risk Driven Tests Systems Tests White Box Testing Black Box Testing	Secure Configuration Secure Deployment	
Other Disciplines	High-Level Risk Assessments		Technical Risk Assessment				Incident Management Patch Management

Focus Perspective: Source Code





```
End Sub
Private Sub tbToolBar_ButtonCl
On Error Resume Next
timTimer.Enabled = True
Select Case Button.Key
    Case "Back"
        brwWebBrowser.Go
    Case "Forward"
        brwWebBrowser.
    Case "Refresh"
        brwWebBrows
    Case "Home"
        WebBr
```

*"In case of software products such as web applications, no matter how you approach the problem of insecure software, from the cost or the engineering perspective, **the majority of security issues are due to coding errors.**"*

Baby Steps in Software Security

What do I tackle first?

Secure Coding Requirements

Describe secure coding requirements in terms of:

1. The common security issues (e.g. OWASP T10)
2. The issue type (e.g. Application Security Frame)
3. The security threat
4. The in-secure code root cause of the vulnerability
5. The "How to" find the vulnerability with black box and white box testing
6. The secure coding requirement/recommendation
7. The risk rating (e.g. STRIDE/DREAD, OWASP)

Common Security Issues: The OWASP Top 10

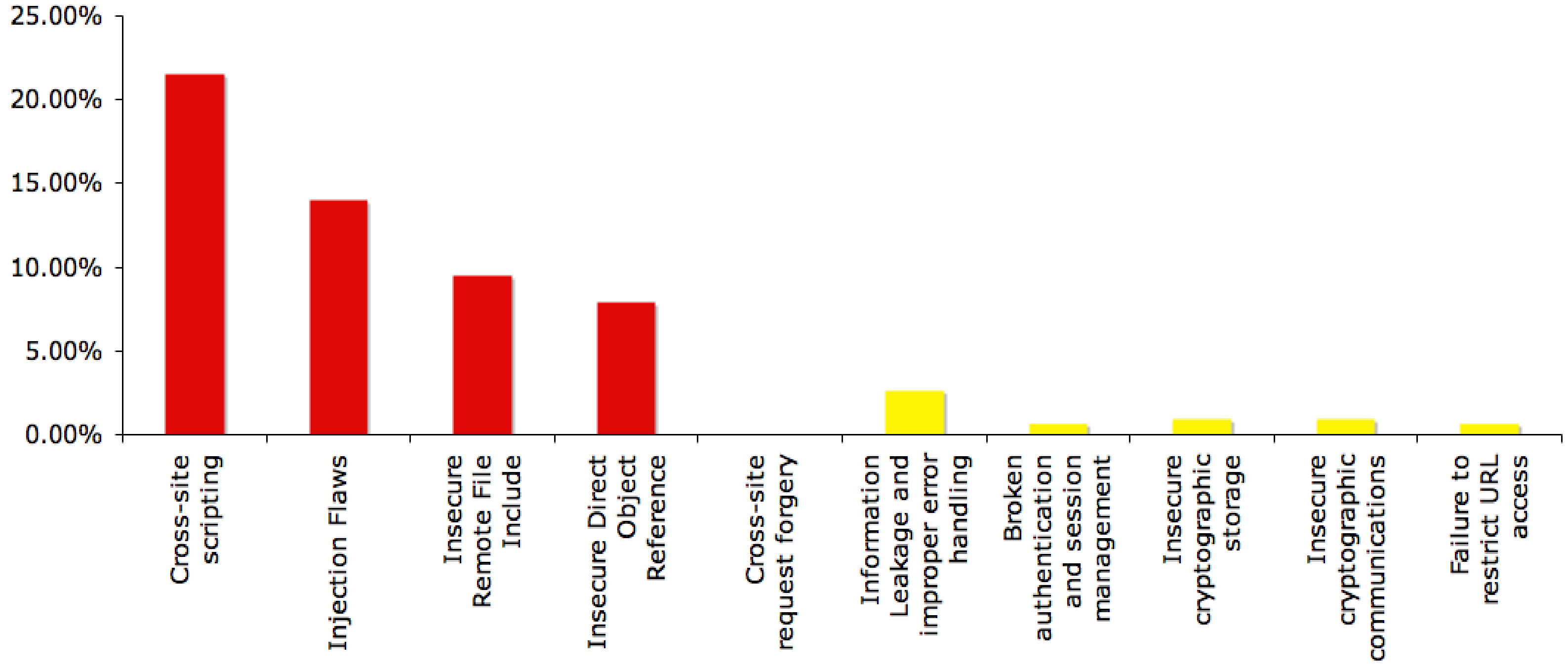
- The Ten Most Critical Issues
- Aimed to educate developers, architects and security practitioners about the consequences of the most common web application security vulnerabilities
- Living document: 2007 T10 different from 2004 T10
- Not a silver bullet for software security
- A great start, but not a standard "per se"

Common Security Issues: OWASP Top 10 2007

1. Cross Site Scripting (XSS)
2. Injection Flaws
3. Insecure Remote File Include
4. Insecure Direct Object Reference
5. Cross Site Request Forgery (CSRF)
6. Information Leakage and Improper Error Handling
7. Broken Authentication and Session Management
8. Insecure Cryptographic Storage
9. Insecure Communications
10. Failure to Restrict URL Access

Common Security Issues: Top 10 Methodology

- Take the MITRE Vulnerability Trends for 2006, and distill the Top 10 *web application security* issues



Common Security Issues: 2007 T10 vs. 2004 T10 what made it and what did not

OWASP Top 10 2007	OWASP Top 10 2004	MITRE 2006 Raw Ranking
1. Cross Site Scripting (XSS)	4. Cross Site Scripting (XSS)	1
2. Injection Flaws	6. Injection Flaws	2
3. Insecure Remote File Include (NEW)		3
4. Insecure Direct Object Reference	2. Broken Access Control (split in 2007 T10)	5
5. Cross Site Request Forgery (CSRF) (NEW)		36
6. Info Leakage and Improper Error Handling	7. Improper Error Handling	6
7. Broken Auth. and Session Management	3. Broken Authentication and Session Management	14
8. Insecure Cryptographic Storage	8. Insecure Storage	8
9. Insecure Communications (NEW)	Discussed under 10	8
10. Failure to Restrict URL Access	2. Broken Access Control (split in 2007 T10)	14
	1. Unvalidated Input	7
	5. Buffer Overflows	4, 8, and 10
	9. Denial of Service	17
	10. Insecure Configuration Management	29

Security Threats and OWASP T10 Vulnerabilities

- **Phishing**
  - Exploit weak authorization, authorization, session management and input validation (XSS, XFS) vulnerabilities
- **Privacy violations**
  - Exploit poor input validation, business rule and weak authorization, injection flaws, information leakage vulnerabilities
- **Identity theft**
  - Exploit poor or non-existent cryptographic controls, malicious file execution, authentication, business rule and auth checks vulnerabilities

Security Threats and OWASP T10 Vulnerabilities (Cont)

- **System compromise, data destruction**
  - Exploit injection flaws, remote file inclusion-upload vulnerabilities
- **Financial loss**
  - Exploit unauthorized transactions and CSRF attacks, broken authentication and session management, insecure object reference, weak authorization-forceful browsing vulnerabilities
- **Reputation loss**
  - Depend on any evidence (not exploitation) of a web application vulnerability

A1: Cross Site Scripting

- **Issue**
  - A web site that gathers user input and reflects input back to the browser
- **Threats**
  - Attacker crafts a URL by attaching to it a malicious script that is sent via phishing or posted as a link on a malicious site. The malicious script executes on the user victim browser

A1: Cross Site Scripting - Insecure Software Root Cause

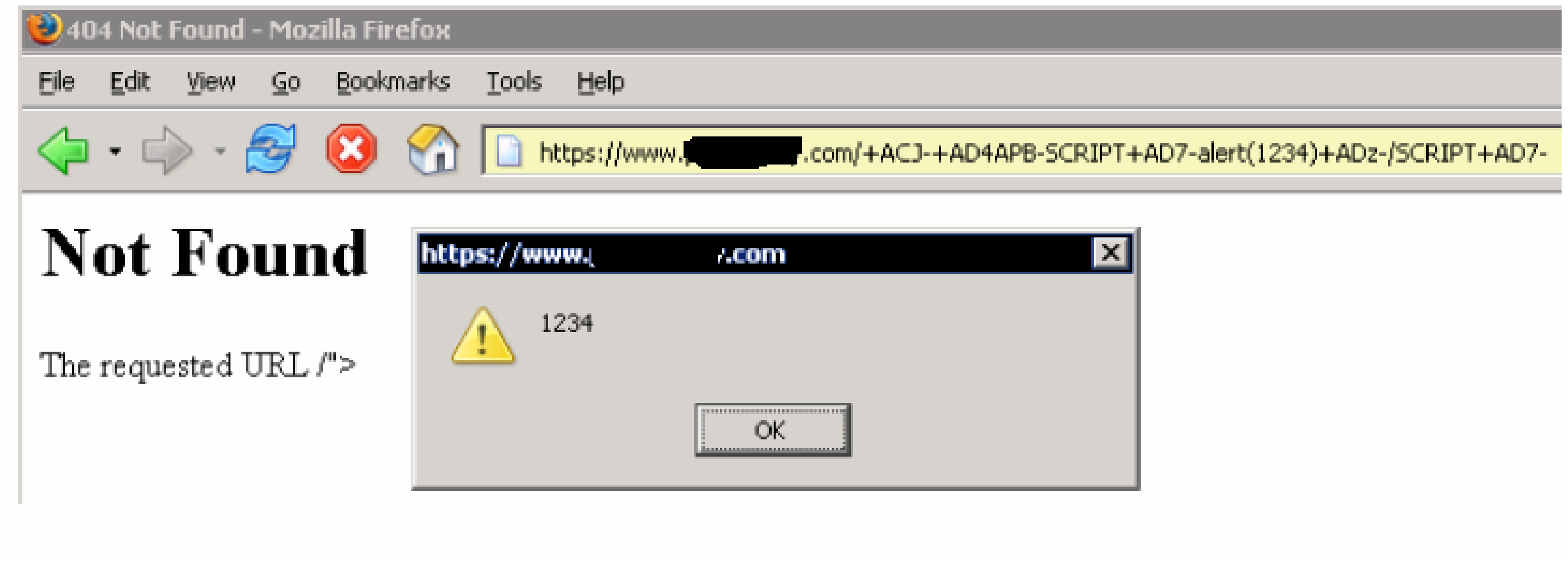
```
import java.io.*;
import javax.servlet.http.*;
import javax.servlet.*;
public class HelloServlet extends HttpServlet
{
public void doGet (HttpServletRequest req,
HttpServletResponse res) throws ServletException,
IOException
{
String input = req.getHeader(“USERINPUT”);
PrintWriter out = res.getWriter();
out.println(input);  // echo User input.
out.close();
}
}
```

A1: Cross Site Scripting -How to find the potential vulnerability

- Verify whether an application or web server will respond to requests containing simple scripts with an HTTP response that are executed by the user's browser.
- The attack vector can be a script to show sensitive information (e.g. cookie stored on the browser) in an alert:

```
http://server/cgibin/testcgi.exe?<SCRIPT>alert("Cookie"+document.cookie)</SCRIPT>
```

A1: Cross Site Scripting -How to find the potential vulnerability


404 Not Found - Mozilla Firefox
File Edit View Go Bookmarks Tools Help

https://www.[redacted].com/+ACJ-+AD4APB-SCRIPT+AD7-alert(1234)+ADz-/SCRIPT+AD7-

Not Found

The requested URL /">

https://www.[redacted].com

1234

Mitigate Encoded XSS Vectors!

A1: Cross Site Scripting - Secure coding requirements

1. **Perform input data validation** using white lists (e,g, default deny) of unsafe characters and output encoding. When using .NET make sure that request validation is enabled as well as HTML encoding for the content to be displayed.
   - ***<pages validateRequest="true" ... />***

     Server.HtmlEncode(string)

A1: Cross Site Scripting - Secure coding requirements

2. **Enforce encoding in output** to assure that the browser interprets any special characters as data and markup. HTML encoding usually means **<** becomes **&lt;**,

   > becomes **&gt;**,

   & becomes **&amp;**, and

   " becomes **&quot**

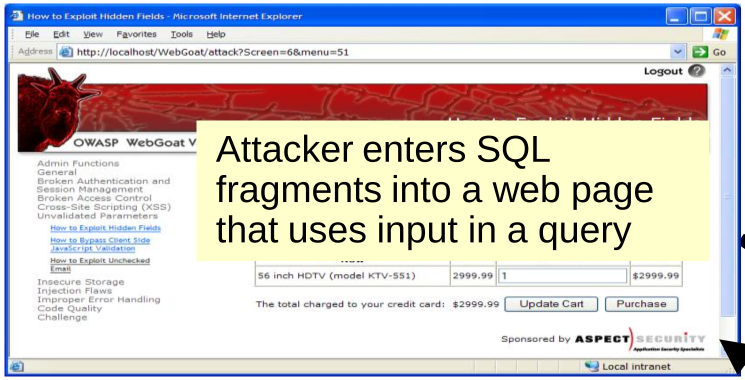
   The text <script> would be displayed as <script> but on viewing the markup it would be represented by **&lt;script&gt;**

A2: Injection Flaws –SQL Injection Example

- **Issue**
  - Unfiltered characters that can be used for SQL commands and use of dynamic queries
- **Threat**
  - Malicious user constructs an input containing malicious SQL query and supplies it in the input variable.
  - The application passes the variable without filtering directly to the dynamically constructed SQL query or stored procedure.
  - SQL malicious query executes on the sever

A2: Injection Flaws –SQL Injection Attack Illustrated

1 Attacker sends data containing SQL fragments



How to Exploit Hidden Fields - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Address http://localhost/WebGoat/attack?Screen=6&menu=51

Logout

OWASP WebGoat V

Admin Functions
General
Broken Authentication and Session Management
Broken Access Control
Cross-Site Scripting (XSS)
Unvalidated Parameters
- How to Exploit Hidden Fields
- How to Bypass Client Side JavaScript Validation
- How to Exploit Unchecked Email

Insecure Storage
Injection Flaws
Improper Error Handling
Code Quality
Challenge

56 inch HDTV (model KTV-551)	2999.99	1	$2999.99

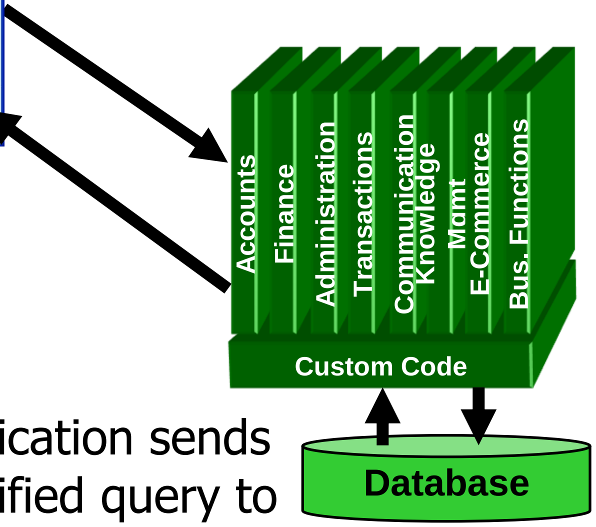
The total charged to your credit card: $2999.99 Update Cart Purchase

Sponsored by ASPECT SECURITY Application Security Specialists

Local intranet

Attacker enters SQL fragments into a web page that uses input in a query

3 Attacker views unauthorized data



2 Application sends modified query to database, which executes it

A2: Injection Flaws-SQL Injection - Insecure Software Root Cause

```
public List getProductsByTitleKeyWords(String[]
keywords)
{
   JdbcTemplate jt = new
JdbcTemplate(getDataSource());

      String query = "select * from products
where "+
   createCriteria(keywords);

      List list = jt.query(query, new

   ProductRowMapper());
      Iterator iter = list.iterator();
         while (iter.hasNext()) {
   Product prod = (Product) iter.next();
       prod.setFeedback(getFeedBacks(prod));
      }
      return list;
}
```

A2: Injection Flaws-SQL SQL Injection– How To Find The Potential Vulnerability

- **Via Penetration Testing Using Attack Vectors**
  - ***'' (double apostrophe), (single quote)***
  - ***, (comma)***
  - ***--' (comment)***
  - ***OR '1'='1—' (always true statement)***
  - ***aaa'; DROP TABLE Docs;-- (use semicolon to break out the query)***
- **Via Source Code Analysis**
  - Looks for instances of dynamic queries constructions that use statement instead of Prepared statements

A2: Injection Flaws-SQL Injection - Secure coding requirements

1. **Use SQL Parameterized Queries** Instead of dynamic SQL generation: ***`SELECT * FROM users WHERE username=?`***
2. **Use stored procedures** to reduce the risk of SQL injection (no SPs with dynamically build queries! you need to pass parameters)
3. **Use prepared statements** such asstrongly typed "PreparedStatement" in .NET use "SqlCommand" with "SqlParameters"

A2: Injection Flaws-SQL Injection - Secure coding requirements

4. **Filter user input** to remove special characters:

```
' "  ` ; * % _
=&\|*?~<>^()[]{}$\n\r
```

5. **Limit write database privileges** for application’s Functional ID (no DROP privileges!)
6. **Avoid detailed error messages** (e.g. SQL Exception Information) that are useful to an attacker

A3: Malicious File Execution

- **Issue**
  - Parameter manipulations leading to command execution
  - Upload function can be used to upload malicious scripts.
- **Threats**
  - Arbitrary commands can be run in the application context by the operating system
  - Malicious files (e.g. script) can be executed on the application server

A3: Malicious File Execution - Insecure Software Root Cause

```
String[] cmdArray = new String[2]; // String array to store command

Runtime runtime = Runtime.getRuntime();
try {
    cmdArray[0] = "cmd.exe /C" ;
    String fromRequest = request.getParameter("cmd");
    cmdArray[1] = "dir \""+fromRequest+"\"";
    Process process = runtime.exec(cmdArray);
}
```

…

A3: Malicious File Execution – How To Find the Vulnerability

- If a user passes the following information in the cmd parameter:

```
cmd=%3B+mkdir+hackerDirectory
```

- At the code level:

```
cmdArray[0] = "cmd.exe /C" ;
        String fromRequest =
"%3B+mkdir+hackerDirectory"
        cmd[1] = "dir \""+fromRequest+"\"";
        Process process = runtime.exec(cmd);
```

- Final command executed is:

```
cmd.exe /C "dir; mkdir hackerDirectory"
```

A3: Malicious File Execution – Secure Coding Requirements

1. **Do not use user controllable input** when executing commands on the operating system
2. **Outline all acceptable values for the user input** (white list) and reject all other values (default deny) before executing the command
3. **Verify if the specified value is appropriate** by using an hash table lookup
4. **Make sure that encoded commands are escaped** before execution

A4: Insecure Direct Object Reference

- **Issue**
  - Invalidated reference to an internal implementation object, such as a file, directory, database record, or key, as a URL or form parameter
- **Threats**
  - An attacker can manipulate direct object references to access other objects without authorization, unless an access control check is in place.

A4: Insecure Direct Object Reference: Insecure Root Causes

- This code can be attacked to access the file system:

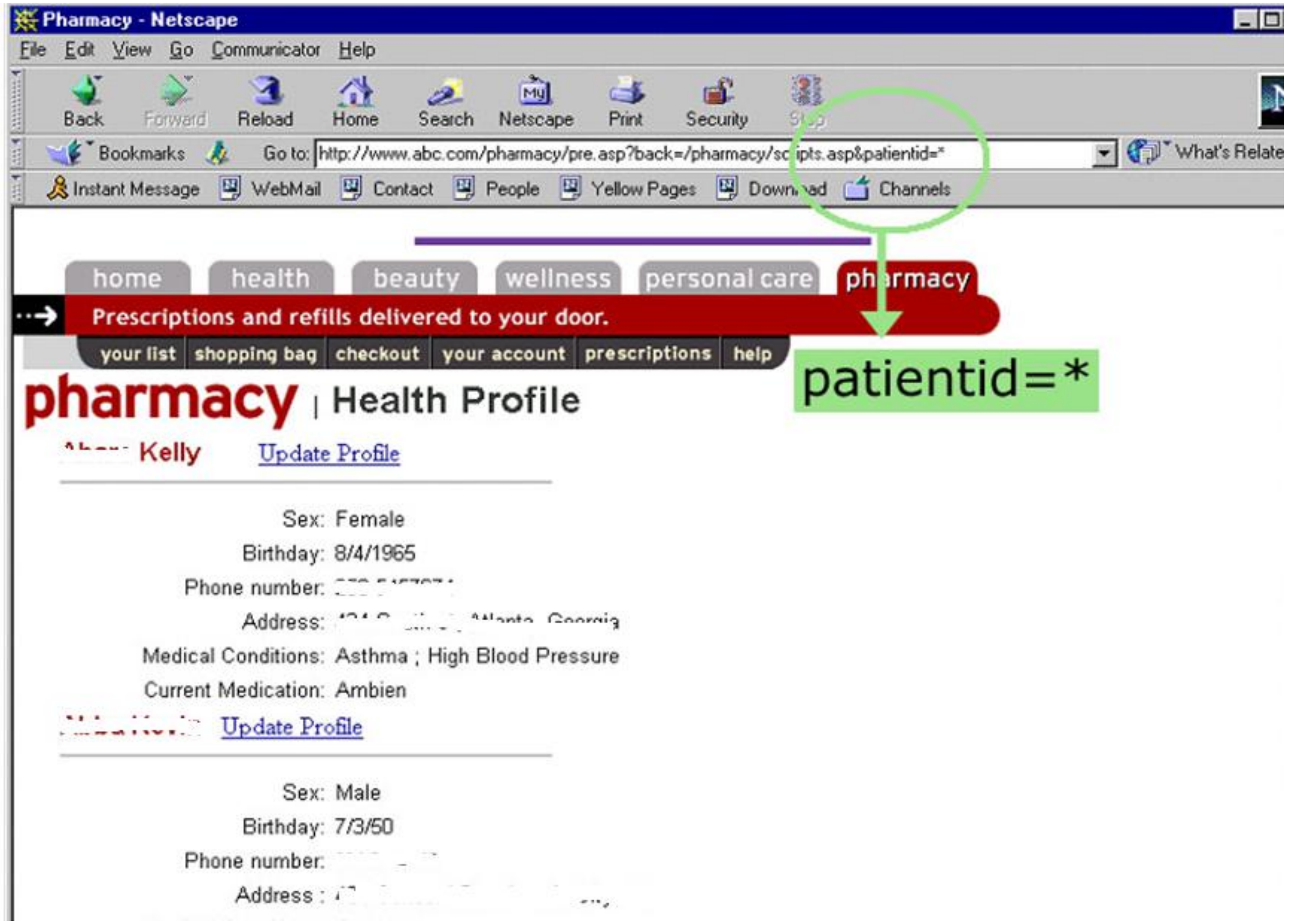
  -
    ```
    <select name="language"><option
    value="fr">Français</option></select>
    ……require_once
     ($_REQUEST['language’]."lang.php");
    ```
- An attacker can change the cartID parameter to whatever cart they want:

  -
    ```
    int cartID = Integer.parseInt(
    request.getParameter( "cartID" ) );
    String query = "SELECT * FROM table
       WHERE cartID=" + cartID;
    ```

A4: Insecure Direct Object Reference: How You Can Find If You Are Vulnerable

- **Black Box Testing**
  - Check if user parameters can be manipulated to access other pages without authorization or to access objects via parameter manipulation
- **White Box Testing**
  - Check that object references to users are validated
  - Check user entitlements to object
  - Check for any trusted user controlled input when specify filenames, paths etc

A4: Insecure Direct Object Reference - How You Can Find If You Are Vulnerable


Pharmacy - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Stop
Bookmarks Go to: What's Related
Instant Message WebMail Contact People Yellow Pages Download Channels
home health beauty wellness personal care pharmacy
your list shopping bag checkout your account prescriptions help

http://www.abc.com/pharmacy/pre.asp?back=/pharmacy/scripts.asp&patientid=*

patientid=*

Prescriptions and refills delivered to your door.

pharmacy | Health Profile

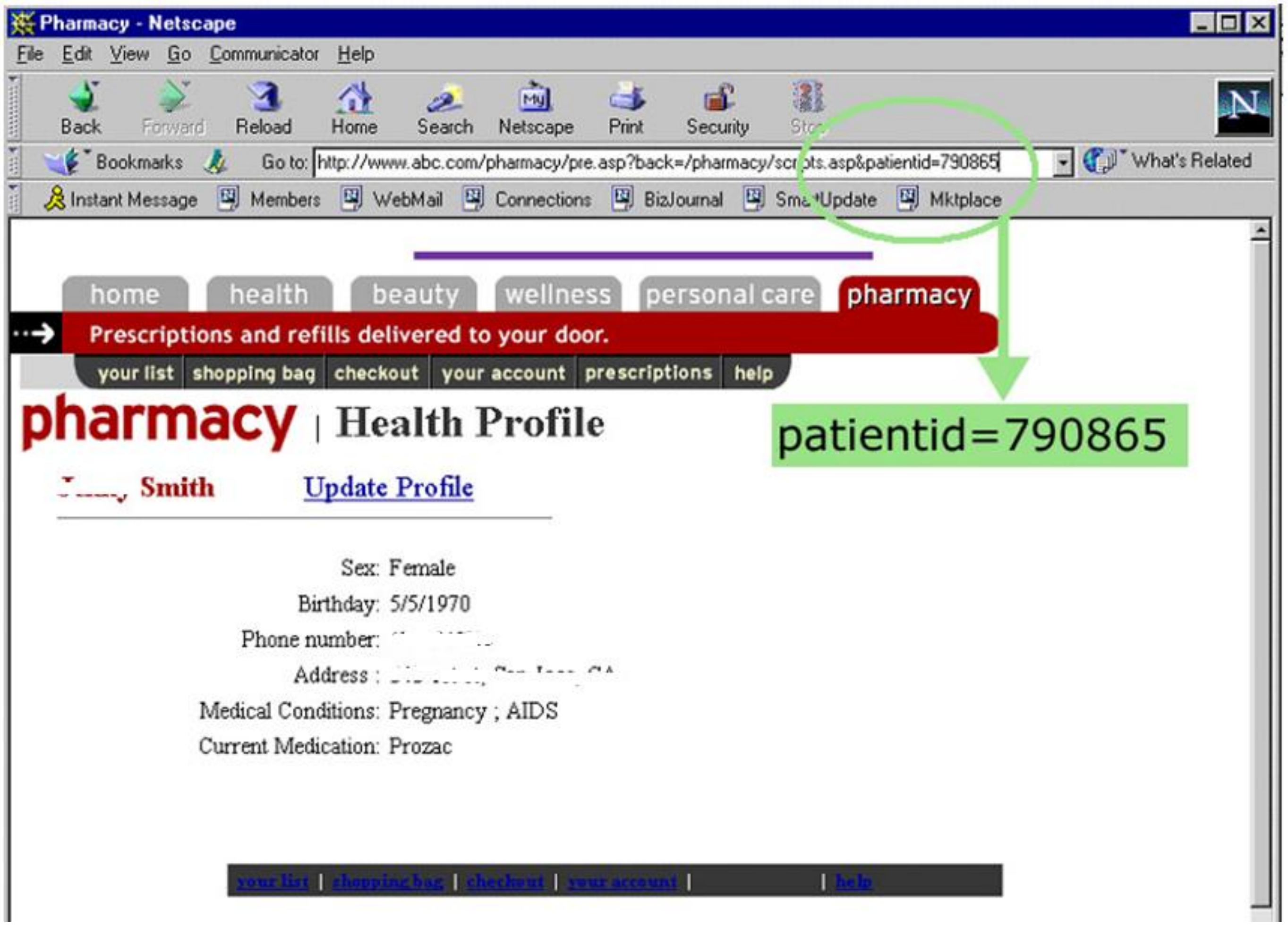
[redacted] Kelly Update Profile

Sex:	Female
Birthday:	8/4/1965
Phone number:	[redacted]
Address:	[redacted] Atlanta, Georgia
Medical Conditions:	Asthma ; High Blood Pressure
Current Medication:	Ambien

[redacted] Update Profile

Sex:	Male
Birthday:	7/3/50
Phone number:	[redacted]
Address :	[redacted]

A4: Insecure Direct Object Reference - How You Can Find If You Are Vulnerable


Pharmacy - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Stop
Bookmarks Go to: What's Related
Instant Message Members WebMail Connections BizJournal SmartUpdate Mktplace
home health beauty wellness personal care pharmacy
your list shopping bag checkout your account prescriptions help

http://www.abc.com/pharmacy/pre.asp?back=/pharmacy/scripts.asp&patientid=790865

patientid=790865

Prescriptions and refills delivered to your door.

pharmacy | Health Profile

[illegible] Smith Update Profile

Sex:	Female
Birthday:	5/5/1970
Phone number:	[illegible]
Address :	[illegible]
Medical Conditions:	Pregnancy ; AIDS
Current Medication:	Prozac

your list | shopping bag | checkout | your account | | help

A4: Insecure Direct Object Reference - Secure Coding Requirements

1. **Avoid exposing direct object references to users by using an index**, indirect reference map, or other indirect method that is easy to validate. If a direct object reference must be used, ensure that the user is authorized before using it.
2. **Avoid exposing your private object references to users** whenever possible, such as primary keys or filenames

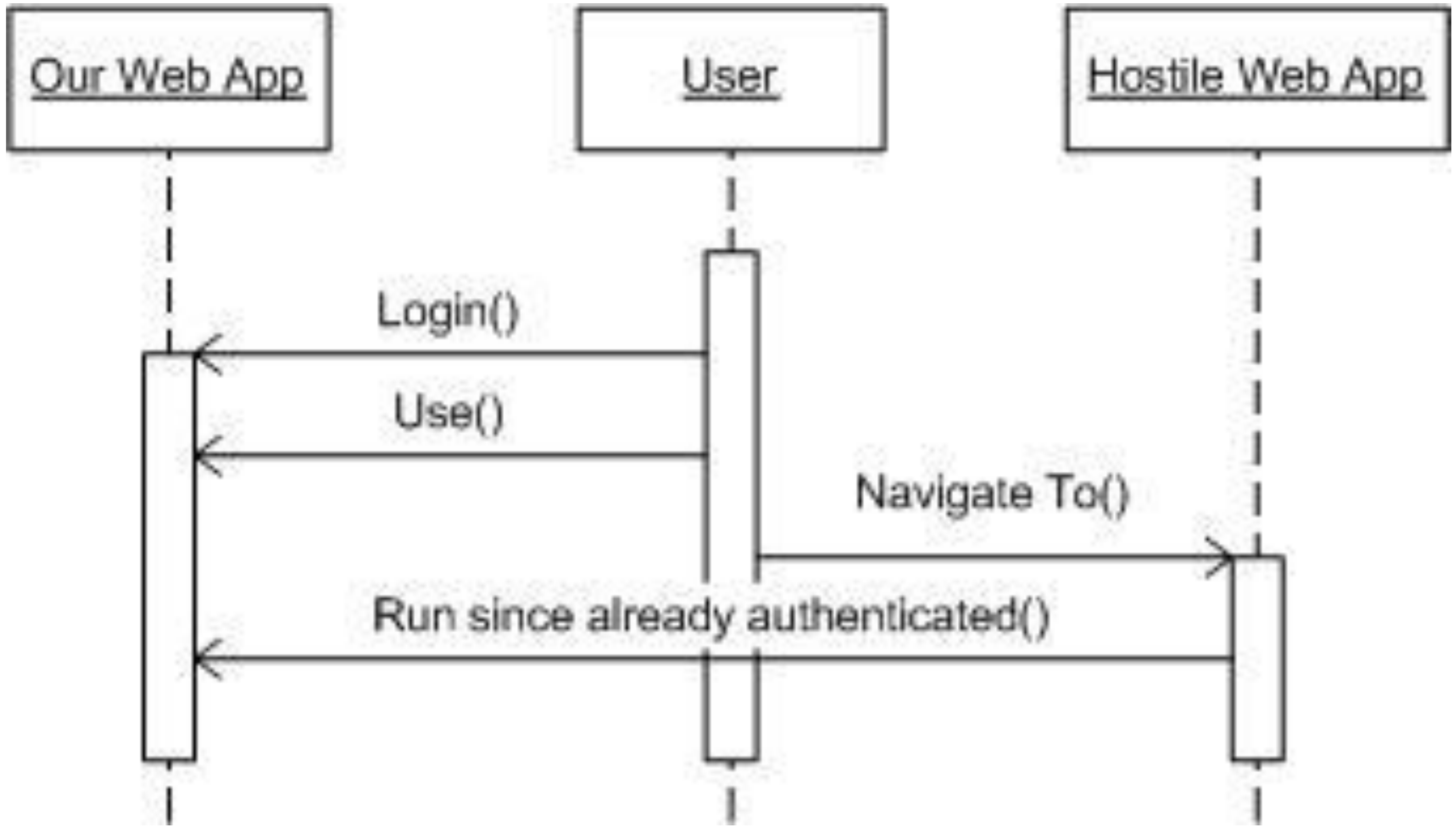
A4: Insecure Direct Object Reference - Secure Coding Requirements

3. **Validate any private object references** extensively with an "accept known good" approach
4. **Verify authorization to all referenced objects** Use an index value or a reference map to prevent parameter manipulation attacks.

A5: Cross Site Request Forgery

- **Issue:**
  - A CSRF attack forces a logged-on victim's browser to send a request to a vulnerable web application, which then performs the chosen action on behalf of the victim. Any web application without a build in CSRF control is vulnerable
- **Threats:**
  - May direct the user to invoke logouts and steal user credentials. In a bank application might invoke processing requests such as transfer of funds.

A5: Cross Site Request Forgery



A5: Cross Site Request Forgery- in-secure software root causes

- Non re-authenticated high risk transactions

```
<img
src="http://www.bank.com/transfer.do?frmA
cct=document.form.frmAcct&
toAcct=4345754&toSWIFTid=434343&amt=3434.
43">
```

- Auto-POST of confidential data

```
<form> <input type="test" name="name"
value="John"/> <input type="test"
name="lastname" value="Dawson"/> <input
type="submit"/> </form>
```

A5: CSRF: How You Can Find If You Are Vulnerable

- Check source code for forms that authorize requests on automatic credentials (session cookies, remember me functionality, SSO tokens)
  - Auto-Posting forms
  - ***`<img>, <iFrame> and <script>`***
  - XMLHTTPRequests
- Some automated scanners can detect CSRF today
- Record and replay transactions, manually check for attack vectors
  - ***`<img src="http://www.example.com/logout.php">`***

A5: CSRF: Secure Coding Requirements

1. **Insert custom random tokens** into every form and URL

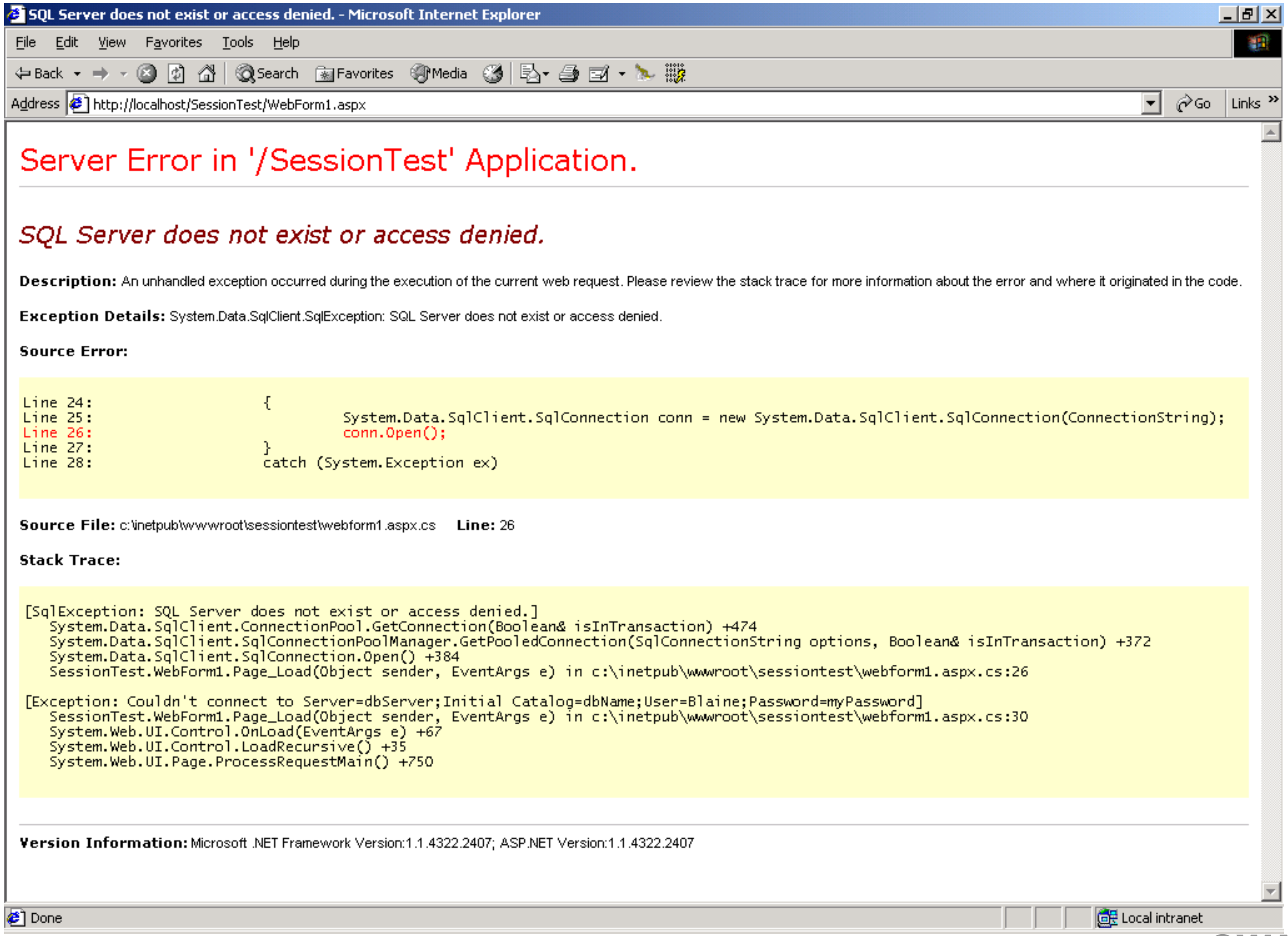
```
<form action="/transfer.do" method="post">
   <input type="hidden" name="8438927730"
   value="43847384383"> … </form>
```

2. **Make sure there a no XSS vulnerabilities**
3. **Re-authenticate and perform out of band verification** when performing high risk transaction
4. **Do not use GET requests for sensitive data** or to perform high risk transactions
5. **For ASP.NET set ViewStateUserKey** (similar check as random token)

A6: Information Leakage and Improper Error Handling

- **Issue**
  - Coding errors in exception handling and error reporting can leak information about the application or the user
- **Threats**
  - Detailed error handling, stack traces in default error messages can disclose application information
  - Non generic error messages
  - Error codes in URL parameters can give insight to validation of user credentials

A6: Information Leakage and Improper Error Handling


SQL Server does not exist or access denied. - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost/SessionTest/WebForm1.aspx
Go Links
Done
Local intranet

Server Error in '/SessionTest' Application.

SQL Server does not exist or access denied.

Description: An unhandled exception occurred during the execution of the current web request. Please review the stack trace for more information about the error and where it originated in the code.

Exception Details: System.Data.SqlClient.SqlException: SQL Server does not exist or access denied.

Source Error:

```
Line 24:                    {
Line 25:                            System.Data.SqlClient.SqlConnection conn = new System.Data.SqlClient.SqlConnection(ConnectionString);
Line 26:                            conn.Open();
Line 27:                    }
Line 28:                    catch (System.Exception ex)
```

Source File: c:\inetpub\wwwroot\sessiontest\webform1.aspx.cs **Line:** 26

Stack Trace:

```
[SqlException: SQL Server does not exist or access denied.]
   System.Data.SqlClient.ConnectionPool.GetConnection(Boolean& isInTransaction) +474
   System.Data.SqlClient.SqlConnectionPoolManager.GetPooledConnection(SqlConnectionString options, Boolean& isInTransaction) +372
   System.Data.SqlClient.SqlConnection.Open() +384
   SessionTest.WebForm1.Page_Load(Object sender, EventArgs e) in c:\inetpub\wwwroot\sessiontest\webform1.aspx.cs:26

[Exception: Couldn't connect to Server=dbServer;Initial Catalog=dbName;User=Blaine;Password=myPassword]
   SessionTest.WebForm1.Page_Load(Object sender, EventArgs e) in c:\inetpub\wwwroot\sessiontest\webform1.aspx.cs:30
   System.Web.UI.Control.OnLoad(EventArgs e) +67
   System.Web.UI.Control.LoadRecursive() +35
   System.Web.UI.Page.ProcessRequestMain() +750
```

Version Information: Microsoft .NET Framework Version:1.1.4322.2407; ASP.NET Version:1.1.4322.2407

A6: Information Leakage and Improper Error Handling- insecure software root cause

- Declarative setting in “web.config” file “customErrors” set to Off and no custom re-direct
  - `<customErrors mode=“Off”/>`
- Error message and stack trace is displayed to the user using “Server.GetLastError().ToString()”

  -
    ```
    <script language="C#" runat="server"> Sub
    Page_Error(Source As Object, E As EventArgs)
    Dim message As String = "<font face=verdana
    color=red><h1>" & Request.Url.ToString()&
    "</h1>" & "<pre><font color='red'>" &
    Server.GetLastError().ToString()&
    "</pre></font>" Response.Write(message) //
    display message End Sub </script>
    ```

A6: Information Leakage and Improper Error Handling- How You Can Find If You Are Vulnerable: Black Box Testing

- Force errors to verify account harvesting vulnerabilities
  - ***"The password you entered was not recognized. Please enter it again***
- Force errors to verify information disclosure via exception handling:
  - ***[SqlException (0x80131904): An error has occurred while establishing a connection to the server.***

A6: Information Leakage and Improper Error Handling- How You Can Find If You Are Vulnerable: White Box Testing

- JAVA
  - Information leakage can occur when developers use ***`printStackTrace()`*** and ***`getStackTrace()`*** exception methods
- .NET
  - Information leakage can occur when developers use objects such as ***`System.Exception`*** with ***`ApplicationException`*** and ***`SystemException`*** and ***`Exception`*** object ***`StackTrace`***

A6: Information Leakage and Improper Error Handling -Secure Coding Requirement

1. **Made exception information only to be used as debugging information** that is not part of production release code. Use `Log4jLogger` to log exception error messages securely
2. **Use declarative programming** setting in “web.config” file and set “customErrors” to On and “mode=RemoteOnly”.
3. **Use centrailized exception handling** (e.g. structsActionMessages & ActionErrors)
4. **Do not display specific errors** that allow for account harvesting

A7: Broken Authentication and Session Management

- **Broken Authentication Issues**
  - Weak or no authentication as well in-secure password management, weak passwords, remember me features and "Autocomplete" set OFF in web forms, weak secret answer combination for password reset
- **Broken Authentication Threats**
  - Flaws can lead to the spoofing of the credentials in transit, man in the middle attacks, brute forcing of password and guessing of passwords

A7:Broken Authentication and Session Management

- **Session Management Issues**
  - Failure to protect credentials and session tokens through their lifecycle. Common issues include:
    - session tokens not re-issued after authentication
    - not marked secure
    - passed in clear
    - passed via GET requests
    - with guessable values
    - remaining active after logout and idle logout.

A7: Broken Authentication and Session Mgmt

- **Session Management Threats**
  - These flaws can lead tobHijacking of user or administrative accounts, undermine authorization and accountability controls and cause privacy violations.

A7: Broken Authentication - insecure software root cause

```
Http Cookie MyCookie;
MyCookie = Request.Cookies  [“CookiesLoginAttempts”];
MyCookie.Expires=now.AddHours(10);
//decrement
int logInAtt=Convert.ToInt32(MyCookie.Value.ToString());
CookieVal=int.Parse (MyCookie.Value.ToString());
If (CookieVal >0)
   CookieVal-=1;
//store in response cookie
HttpCookie AttemptCntCookie = new HttpCookie (“CookieLoginAttempts”);
AttemptCntCookie.Value =CookieVal.ToString();
```

A7: Session Management - insecure software root cause

- **Cookies with confidential information**

1. *`Set-Cookie: userid=jdoe; expires=Thu, 01-Jun-2006 19:16:08 GMT; path=/`*
2. *`Set-Cookie: password=xxxxxxx; expires=Thu, 01-Jun-2006 19:16:08 GMT; path=/`*

- **Cookies not marked Secure**
  - *Set-Cookie: name=newvalue; expires=date; path=/; domain=.example.org.*

A7: Broken Authentication - How You Can Find If You Are Vulnerable

- **Automated Tools**
  - Can only identify use of weak authentication such as basic authentication, “Autocomplete” OFF in web forms and remember me functionality
- **Manual Test**
  - To identify weak passwords (complexity), flaws in password reset/change, timeouts and logoff functionality, use of SSL
- **Source Code Analysis**
  - Authentication setting in configuration files (e.g. authentication mode="Forms“)

A7: Session Management - How You Can Find If You Are Vulnerable

- **Automated Tools**
  - Most of automated scanning tool can only identify session cookies not set with secure flag, passed via GET instead of POST and unpredictability (e.g. CookieDigger)
- **Manual Ethical Hacking with Web Proxy**
  - Best way to find weak session management, session invalidation at logout and re-issuance **after authentication**
- **Source Code Analysis**
  - For flaws in user and session management

A7: Session Management -Secure Coding Requirements

- **Session Management**
  1. Consider using sessionID and manage session on the server.
  2. Invalidate the existing Session ID before authentication
  3. Issue a new Session ID after authentication
  4. Invalidate this Session ID on logout
  5. Set secure flag defaults to TRUE
  6. Pass session IDs in secure cookies instead of in URL parameters
  7. Use POST instead of GET when passing sensitive parameters
  8. Should be random (128 bit)

A7: Broken Authentication -Secure Coding Requirements

- **Authentication**
  1. Do not use weak form authentication such as BASIC or NTLM
  2. Ensure that SSL is used
  3. Ensure that logins start with an encrypted web page
  4. Ensure that logouts are available in every page
  5. Use only shared secrets in challenge/responses
  6. Use trusted authentication (e.g. SSO) not impersonation
  7. Implement idle time-out

A7: Broken Authentication -Secure Coding Requirements

- **Passwords**
  1. Enforce password complexity
  2. Require old passwords for setting new
  3. Use challenge/response and out of band for re-setting passwords
  4. Store passwords with irreversible encryption

A8: Insecure Cryptographic Storage

- **Issues**
  - Failing to protecting sensitive data with cryptography Failing to encrypt sensitive data because of either using weak encryption algorithms or short encryption keys.
  - Home-grown encryption
  - Failure to protect secrets such as private keys via hard-coding and unprotected access

A8: Insecure Cryptographic Storage

- **Threats**
  - Disclosure of customer sensitive information,
  - Exposure of authentication data to unauthorized users
  - Exposure of secrets such as keys and challenge response answers

A8: Insecure Cryptographic Storage - Insecure software root cause

- **Hard-coding of passwords**

```
int VerifyPwd(String password) { if
   (passwd.Equals("68af404b513073584c4b6f2
   2b6c63e6b")) { } return(0) return(1);}
```

- **Errors when coding cryptography**

```
public static String digest(String
password) {
MessageDigest md5
=MessageDigest.getInstance("MD5");
byte[] hash =
md5.update(password.getBytes());
return makeStringFromBytes(hash);}
```

A8: Insecure Cryptographic Storage - How You Can Find If You Are Vulnerable

- **Automated Source Code Analysis**
  - Can verify instances of use of unsafe algorithms (MD5, DES, SHA1) as well as hard-coded keys and credentials
- **Manual Source Code Analysis**
  - Home grown cryptography such as missing to use salt and seed when using digests
  - key sizes
  - Failing to use encryption for sensitive data and authentication data
  - Weak keys and shared secrets management.

A8: Insecure Cryptographic Storage -Secure Coding Requirements

1. **Use approved algorithms**

   (e.g. AES, RSA, SHA-256 instead of Blowfish, RC4, SHA1, MD5) and recommended key strength (128 bit for symmetric and 1048 for public)

2. **Encrypt authentication credentials in storage and transit**
3. **Protect PII and customer sensitive data in storage and transit as appropriate**

   Do not store credit card data (CVV2, magnetic strip information) see PCI compliance

4. **Store keys in secure repositories**

   Use HSM and secure key storage such as CryptoAPI or Java Key Store

A9: Insecure Communication

- **Issues**
  - Failure to encrypt network traffic to protect sensitive communications.
  - Not using SSL for communication with end users as well as the back-end.
- **Threats**
  - Identity theft, financial fraud
  - Non-compliance with standards
  - Loss of sensitive data such as credit card information, bank account information and health care information

A9 Insecure Communication - insecure software root cause

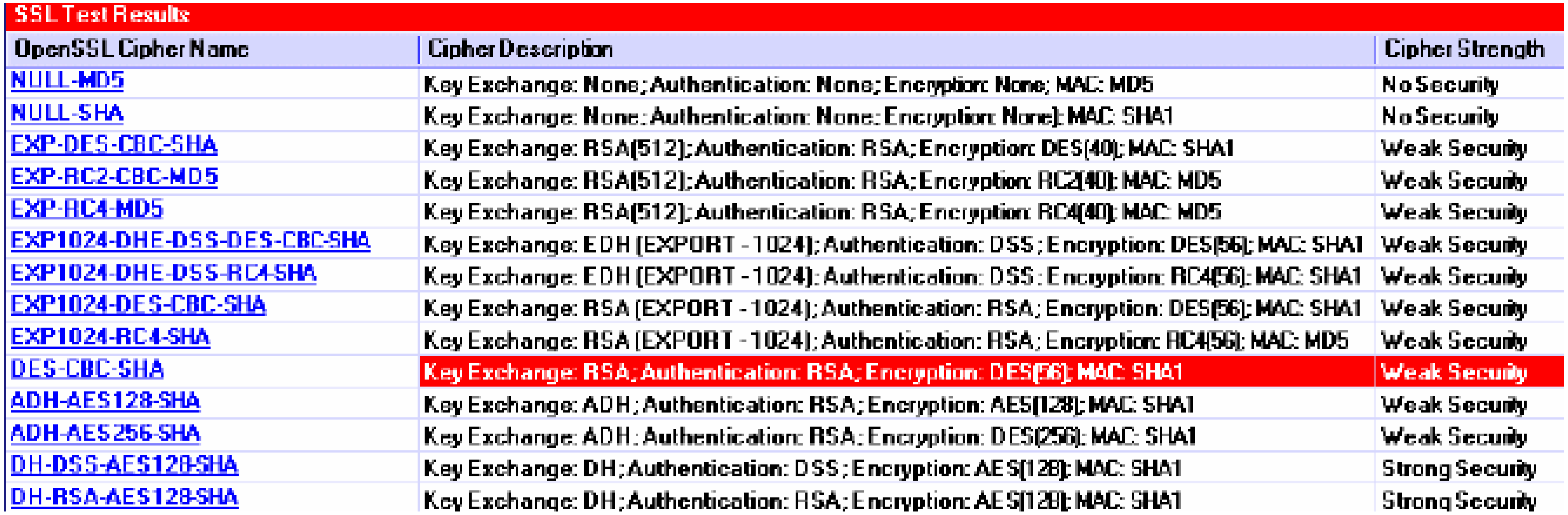
- Lack of configuration of SSL on the web server secure connection properties are not set to true. Tomcat 3.3. default configuration example:

```
<Http10Connector port="8080"
    secure="false"
    maxThreads="100"
    maxSpareThreads="50"
    minSpareThreads="10" />
```

A9 Insecure Communication - How You Can Find If You Are Vulnerable

- **Vulnerability scanning tools**
  - Can verify that SSL is used
  - Can only verify front end not back end
  - Foundstone SSL digger can verify encryption strength (strong chipers enabled)
- **Manual Test**
  - Old browsers such as Netscape 7.2 have custom setting for SSL that can be used to verify SSL configuration
- **Code Review**
  - Can verify use of API that enable SSL connection in the back end (e.g. middle-tier and database, directories)

A9: Insecure Communication
- How You Can Find If You Are Vulnerable



SSL Test Results

OpenSSL Cipher Name	Cipher Description	Cipher Strength
NULL-MD5	Key Exchange: None; Authentication: None; Encryption: None; MAC: MD5	No Security
NULL-SHA	Key Exchange: None; Authentication: None; Encryption: None]; MAC: SHA1	No Security
EXP-DES-CBC-SHA	Key Exchange: RSA(512); Authentication: RSA; Encryption: DES(40); MAC: SHA1	Weak Security
EXP-RC2-CBC-MD5	Key Exchange: RSA(512); Authentication: RSA; Encryption: RC2(40); MAC: MD5	Weak Security
EXP-RC4-MD5	Key Exchange: RSA(512); Authentication: RSA; Encryption: RC4(40); MAC: MD5	Weak Security
EXP1024-DHE-DSS-DES-CBC-SHA	Key Exchange: EDH (EXPORT - 1024); Authentication: DSS; Encryption: DES(56); MAC: SHA1	Weak Security
EXP1024-DHE-DSS-RC4-SHA	Key Exchange: EDH (EXPORT - 1024); Authentication: DSS; Encryption: RC4(56); MAC: SHA1	Weak Security
EXP1024-DES-CBC-SHA	Key Exchange: RSA (EXPORT - 1024); Authentication: RSA; Encryption: DES(56); MAC: SHA1	Weak Security
EXP1024-RC4-SHA	Key Exchange: RSA (EXPORT - 1024); Authentication: RSA; Encryption: RC4(56); MAC: MD5	Weak Security
DES-CBC-SHA	Key Exchange: RSA; Authentication: RSA; Encryption: DES(56); MAC: SHA1	Weak Security
ADH-AES128-SHA	Key Exchange: ADH; Authentication: RSA; Encryption: AES(128); MAC: SHA1	Weak Security
ADH-AES256-SHA	Key Exchange: ADH; Authentication: RSA; Encryption: DES(256); MAC: SHA1	Weak Security
DH-DSS-AES128-SHA	Key Exchange: DH; Authentication: DSS; Encryption: AES(128); MAC: SHA1	Strong Security
DH-RSA-AES128-SHA	Key Exchange: DH; Authentication: RSA; Encryption: AES(128); MAC: SHA1	Strong Security

OWASP

A9: Insecure Communication -Secure Coding Requirements

1. **Use SSL**
   - For all connections that are authenticated
   - When transmitting credentials, credit card details, health and other private information
2. **Use transport layer and link layer security**
   - Between web servers and application servers and back end systems and repositories
3. **Address PCI compliance and privacy**

   You much protect credit card holder data and PII in storage and in transit

A10: Failure to restrict URL access

- **Issues**
  - URL web page access is enforced via security by obscurity
  - Failure to enforce users role base access controls to limit access to web pages
- **Threats**
  - A motivated, skilled, or just plain lucky attacker may be able to predict the location of web pages and access these pages, invoke functions, and view data.

A10: Failure to restrict URL access - insecure software root cause

- Typically this is a security flaw, server side RBAC to set which web pages a user should be given access to:

```
if (sess.getCurrentUser().NormalUser ()) {
URLList.add(“View Customer Details",
        "/jsp/Customer.do?action=view&id=“ + custId));

} Else {//must be a super user
    URLList.add(“View Customer Details",
        "/jsp/Customer.do?action=view&id=“ + custId));
    URLList.add(“Edit Customer Details",
        "/jsp/Customer.do? action=edit&id=“ + custId));
    URLList.add(“Delete Customer",
        "/jsp/Customer.do?action=delete&id=“ + custId));
}
```

A10: Failure to restrict URL access - How You Can Find If You Are Vulnerable

- **Automated approaches**
  - A scanning tool, like Nikto has the ability to search for existent files and directories based on a database of well-know resources
  - Static analysis tools are not contextual based and cannot find access controls in the code and link the presentation layer with the business logic.

A10: Failure to restrict URL access - How You Can Find If You Are Vulnerable

- **Manual approaches**
  - Verify the access control mechanism via source code analysis and penetration test. By logging on as user and super user/admin and by forcing access to different web pages can verify that RBAC is enforced.

A10: Failure to restrict URL access -Secure Coding Requirements

1. **Enforce Role Base Access Controls**

   Ensure that RBAC is enforced on the server side to enforce which user has access to which web page

2. **Do not use security by obscurity**

   No HIDDEN parameters to enforce which web pages are accessible

3. **Enforce white list filtering to which web pages should be accessible**

   Only allow file types that you intend to serve, block any attempts to access log files, xml files, etc.

Thank You For Listening

Further Reading

"Web Application Vulnerabilities And Insecure Software Root Causes: Solving The Security Problem From An Information Security Perspective", In-secure magazine, 2008 February Issue: **http://www.netsecurity.org/dl/insecure/INSECURE-Mag-15.pdf**