TOP WEB APPLICATION SECURITY ISSUES

Sharecase 2012 – Patrick Burke
• Core of web security issues is trusting input from somewhere when you should not
OWASP TOP 10 WEB APPLICATION SECURITY RISKS

- A1: Injection
- A2: Cross-Site Scripting (XSS)
- A3: Broken Authentication and Session Management
- A4: Insecure Direct Object References
- 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
• SQL Injection
SQL Injection:

• Parameter passed to database

• Embedding SQL in parameter

• Confusing the database about what is data and what is SQL

• SQL in parameter is executed by database
SQL – Examples:

• SELECT customerID, Companyname, Contactname, Country
  FROM Customers
  WHERE country='Canada'

SQL comments, stuff the database ignores:

• SELECT customerID, Companyname, Contactname, Country
  FROM Customers
  WHERE country='Canada' -- this is a comment...
Choose Country Canada

Show customers
<table>
<thead>
<tr>
<th>Companyname</th>
<th>Contactname</th>
<th>Country</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bottom-Dollar Markets</td>
<td>Elizabeth Lincoln</td>
<td>Canada</td>
</tr>
<tr>
<td>Laughing Bacchus Wine Cellars</td>
<td>Yoshi Tanimura</td>
<td>Canada</td>
</tr>
<tr>
<td>Mère Paillarde</td>
<td>Jean Fresnière</td>
<td>Canada</td>
</tr>
</tbody>
</table>
What happens if we add a quote to the end of the URI?

http://localhost//asp/form_get_output.asp?country=Canada '

  • ' is added to value of the country parameter

The database sees:

  • SELECT customerId, Companyname, Contactname, Country
    FROM Customers
    WHERE country='Canada' '

  • This is bad SQL and the app gets an exception
Error message shows information and data you want:

Microsoft OLE DB Provider for ODBC Drivers error 80040e14

[Microsoft][ODBC SQL Server Driver][SQL Server]Unclosed quotation mark after the character string 'Belgium'.
/test/asp/form_get_output.asp, line 83
What happens when we change the URI from:
  • http://localhost/test/asp/form_get_output.asp?country=Canada
To:
  • Canada' or 1=1 --

Database sees:
  • SELECT customerID, Companyname, Contactname,Country
    FROM Customers
    WHERE country='Canada' or 1=1 --'
### Select Example Output

SQL Query: `SELECT customerID, Companyname, Contactname, Country FROM Customers WHERE country='Canada' or 1=1` -

<table>
<thead>
<tr>
<th>Companyname</th>
<th>Contactname</th>
<th>Country</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ana Trujillo Emparedados y helados</td>
<td>Ana Trujillo Smith</td>
<td>Mexico</td>
</tr>
<tr>
<td>Antonio Morono Taquería</td>
<td>Antonio Moreno</td>
<td>Mexico</td>
</tr>
<tr>
<td>Around the Horn</td>
<td>Thomas Hardy</td>
<td>UK</td>
</tr>
<tr>
<td>Berglunds snabbköp</td>
<td>Christina Berglund</td>
<td>Sweden</td>
</tr>
<tr>
<td>Blauer See Delikatessen</td>
<td>Hanna Moos</td>
<td>Germany</td>
</tr>
<tr>
<td>Blondel père et fils</td>
<td>Frédérique Citeaux</td>
<td>France</td>
</tr>
<tr>
<td>BÁFARÁÁvido Comidas preparadas</td>
<td>Martínez Án Sommer</td>
<td>Spain</td>
</tr>
<tr>
<td>Bon app'</td>
<td>Laurence LeBlanc</td>
<td>France</td>
</tr>
<tr>
<td>Bottom-Dollar Markets</td>
<td>Elizabeth Lincoln</td>
<td>Canada</td>
</tr>
<tr>
<td>B's Beverages</td>
<td>Victoria Ashworth</td>
<td>UK</td>
</tr>
<tr>
<td>Cactus Comidas para llevar</td>
<td>Patricio Simpson</td>
<td>Argentina</td>
</tr>
<tr>
<td>Centro comercial Móctezuma</td>
<td>Francisco Chang Lee</td>
<td>Mexico</td>
</tr>
<tr>
<td>Chop-suey Chinese</td>
<td>Yang Wang</td>
<td>Switzerland</td>
</tr>
<tr>
<td>Comércio Mineiro</td>
<td>Pedro Afonso</td>
<td>Brazil</td>
</tr>
<tr>
<td>Consolidated Holdings</td>
<td>Elizabeth Brown</td>
<td>UK</td>
</tr>
<tr>
<td>Drachenblut Delikatessen</td>
<td>Gven Ottlieb</td>
<td>Germany</td>
</tr>
<tr>
<td>Du monde entier</td>
<td>Janine Labrune</td>
<td>France</td>
</tr>
<tr>
<td>Eastern Connection</td>
<td>Ann Devon</td>
<td>UK</td>
</tr>
<tr>
<td>Ernst Handel</td>
<td>Roland Mendel</td>
<td>Austria</td>
</tr>
<tr>
<td>Familia Arquibaldo</td>
<td>Ana Cruz</td>
<td>Brazil</td>
</tr>
<tr>
<td>FISSA Embra Inter. Salchichas S.A</td>
<td>Diego Pohl</td>
<td>Spain</td>
</tr>
</tbody>
</table>
What is the problem?

• The problem is that data and code are mixed
• The database has no way of knowing that the SQL is being mixed with data

Can you separate the SQL from data?

• Yes - Prepared Statements - separate data from SQL so you avoid having the database parse the data and execute it
$stmt = $dbh->prepare("SELECT * FROM REGISTRY where name = ?");

if ($stmt->execute(array($_GET['name']))) {
    while ($row = $stmt->fetch()) {
        print_r($row);
    }
}
?>
ASP Prepared statement using ADOdb:

Set cmd = Server.CreateObject("ADODB.Command")
Set cmd.ActiveConnection = oConn
Set oRS = Server.CreateObject("ADODB.Recordset")

strSQL = "select * from recommendations where hash_id =? AND confirmation_id =?"
    cmd.CommandText = strSQL
    cmd.CommandType = adCmdText
    cmd.Parameters(0) = Session("recommendations_ID")
    cmd.Parameters(1) = Session("confirmation_id")
set oRS = cmd.execute
SQLI - COUNTER MEASURES:

- Parameterized Query – Prepared Statement – Magic Bullet
- Limit database access. Do not allow open internet access on DB Server
- Reduce account privilege. Use web application role accounts with reduced table and row permissions (no dba or sa accounts for web apps)
- Input Filtering (‘ ; “ SELECT, FROM …) isn't a comprehensive solution
- IPS – Intrusion Prevention System (must be configured to see signs of SQLI & irregular outbound db access) isn't a comprehensive solution
• XSS Cross Site Scripting
• XSS - Cross Site Scripting
• Attackers inject code into a page usually (HTML/JavaScript)
• Victim views page
• Browser believes code is part of site and malicious code executes in the victims browser - bypassing the Same Origin Policy restrictions
First Name: ________ Last Name: ________

Gender: Male ☐ Female ☐

Select your favorite time of day to sleep:

AM ☐ PM ☐

Please choose favorite food: Steak: ☐ Pizza: ☐ Chicken: ☐

Enter your favorite quote:

Select a Level of Education: High School ☐
First Name: Patrick  Last Name: Burke

Gender: Male: ☐  Female: ☐

Select your favorite time of day to sleep: AM, PM

Please choose favorite food: Steak: ☐  Pizza: ☑  Chicken: ☐

This a new XSS Link to a bad website. Please click on me:

Select a Level of Education: College
Hello, Patrick Burke.
You are Male, and you like Pizza
This a new XSS Link to a bad website. Please click on me
You're favorite time is Night, and you passed College!
WHY DOES THIS HAPPEN?

• Think of XSS like SQL injection. Getting code to run that wasn’t meant to be run.

• If you look at an HTML page like a template, with slots where a developer is allowed to put untrusted data. These slots cover the vast majority of the common places where a developer might want to put untrusted data. Putting untrusted data in other places in the HTML is not allowed. This is a "whitelist" model, that denies everything that is not specifically allowed.

• Given the way browsers parse HTML, each of the different types of slots has slightly different security rules. When you put untrusted data into these slots, you need to take certain steps to make sure that the data does not break out of that slot into a context that allows code execution. In a way, this approach treats an HTML document like a parameterized database query - the data is kept in specific places and is isolated from code contexts with escaping.
TERMS:

- **Encoding** describes how the file's characters are physically written in binary (as in Unicode or ANSI).

- **Escaping** refers to the process of replacing special characters (such as `<` and `>`) with their XML entity equivalent (such as `&lt;` and `&gt;`). For URLs, escaping refers to replacing characters with strings starting with `%`, such as `%20` for a single whitespace.

- Escaping differs by language, but encodings are usually widely-accepted standards. Sometimes the terms are used ambiguously (particularly with encoding used to mean escaping), but they are well defined and distinct.
XSS - COUNTER MEASURES:

- Output encoding comes to the rescue. Before output all characters are mapped to another representation so the browser knows to display them on screen instead of executing them.

- Input validation get rid of all the noise and reduce overall security exposure but input validation is hard, so hard that it is impossible to completely filter out XSS.

- Be sure to hide server admin access. This is how Twitter became a victim
XSS - COUNTER MEASURES: OUTPUT ENCODING

• Output encoding: Set the page encoding using meta tag, this ensuring the browser is displaying the encoding you intend to deal with. The actual encoding is context specific, you have to do the right encoding for different part of the HTML document. The OWASP XSS Prevention Cheat Sheet has detailed instruction on what to do where in the HTML page. Various development frameworks have different functions or API to handle the output encoding process.

  • http://www.owasp.org/index.php/XSS_%28Cross_Site_Scripting%29_Prevention_Cheat_Sheet

  • OWASP ESAPI project has created an escaping library in a variety of languages including Java, PHP, Classic ASP, Cold Fusion, Python, and Haskell. Microsoft provides an encoding library named the “Microsoft Anti-Cross Site Scripting Library” for the .NET platform.
XSS - COUNTER MEASURES: INPUT VALIDATION

- **Sanitize with Whitelist**
- Any characters which are not part of an approved list can be removed, encoded or replaced.
- Phone number example: Strip out all non-digit characters.
  - "(555)123-1234", "555.123.1234", and "555\";DROP TABLE USER;--123.1234" all convert to 5551231234.
- Comment form text example: it is difficult to decide on a legitimate set of characters because nearly every character has a legitimate use. One solution is to replace all non alphanumeric characters with an encoded version.
  - So "I like your web page", might emerge from your sanitation routines as "I+like+your+web+page%21". (This example uses URL encoding.)
- File upload/download example: In this case validation is impossible because there is no valid or invalid content. Because your only concern is protecting your app from malicious input and you don't need to actually do anything except accept, store and transmit the file, you can encode the entire file in, say base 64.
XSS - COUNTER MEASURES: INPUT VALIDATION

- **Sanitize with Blacklist**

- Eliminate or translate characters (such as to HTML entities or to remove quotes) in an effort to make the input "safe". As most fields have a particular grammar, it is simpler, faster, and more secure to simply validate a single correct positive test than to try to include complex and slow sanitization routines for all current and future attacks.

```java
public String quoteApostrophe(String input) {
    if (input != null)
        return input.replaceAll("[\']", "&rsquo;");
    else
        return null;
}
```
• Broken Authentication & Session Management
KEY ISSUES

• Problem - Very difficult to write your own authentication management.
• Password hashes and encryption
SENARIO:

• The password database uses unsalted hashes to store everyone’s passwords. A file upload flaw allows an attacker to retrieve the password file. All the unsalted hashes can be brute forced in 4 weeks, while properly salted hashes would have taken years.
SOLUCIONES

• SSO - Campus Single Sign On can help wrap your web applications in an authentication system you don't need to manage.

• Remove need for authentication. Do users need to authenticate. Are they really processing data that can't be seen.

• When SSO doesn't work - use best authentication mechanism possible provided by vendor or web server environment. Be sure to follow all implementation rules closely.

• Encrypt data in transit. using SSL – Campus offers InCommon SSL Certificates for free.
• Cross-Site Request Forgery (CSRF)
CROSS-SITE REQUEST FORGERY (CSRF)

- Cross-Site Request Forgery (CSRF) is a type of attack that occurs when a malicious Web site, email, blog, instant message, or program causes a user’s Web browser to perform an unwanted action on a trusted site for which the user is currently authenticated. The impact of a successful cross-site request forgery attack is limited to the capabilities exposed by the vulnerable application.

- For example, this attack could result in a transfer of funds, changing a password, or purchasing an item in the user's context. In affect, CSRF attacks are used by an attacker to make a target system perform a function (funds Transfer, form submission etc.) via the target's browser without knowledge of the target user, at least until the unauthorized function has been committed.
CSRF - COUNTER MEASURES:

• Developers are encouraged to adopt the Synchronizer Token Pattern (http://www.corej2eepatterns.com/Design/PresoDesign.html). The synchronizer token pattern requires the generating of random "challenge" tokens that are associated with the user's current session.

• There are CSRF prevention modules available for J2EE, .Net, and PHP.

• Challenge-Response is another defense option for CSRF. Examples of challenge-response options (CAPTCHA, Re-Authentication (password), One-time Token).

• The HTTPOnly (https://www.owasp.org/index.php/HTTPOnly) cookie flag was developed specifically to cut down on this risk of stolen session cookies. However, keep in mind that it only cut down on the risk but is not an all encompassing solution. See - http://www.gnucitizen.org/blog/why-httponly-wont-protect-you/ for more information.
PROGRAMMER NOTES:

- Look at www.owasp.org
  - Note the Prevention Measures That Do Not Work
- Look at General Recommendation: Synchronizer Token Pattern.
- If you are .NET programmer pay close attention to proper View State Implementation: (encryption vs hash) (no sensitive data in viewstate)
- Look at Prevention Frameworks for your code base.
- Look closely at Checking Referer Header.
- Do not store sensitive data in include files or cookies
- Review server logs for signs of attack – If attackers are unnoticed they will continue to forge attacks until they get it right.
USER NOTES (NOT TO BECOME A VICTIM):

- Logoff immediately after using a Web application
- Do not allow your browser to save username/passwords, and do not allow sites to “remember” your login
- Do not use the same browser to access sensitive applications and to surf the Internet freely (tabbed browsing).
- The use of plugins such as No-Script makes POST based CSRF vulnerabilities difficult to exploit. This is because JavaScript is used to automatically submit the form when the exploit is loaded. Without JavaScript the attacker would have to trick the user into submitting the form manually.
• Tools
WAF - WEB APPLICATION FIREWALL:

- ModSecurity:
- Supports: Apache and IIS:
- Negative Security Model (Black Listing) - looks for known bad, malicious requests
- Positive Security Model (White Listing) - When positive security model is deployed, only requests that are known to be valid are accepted, with everything else rejected.
- Virtual Patching (code patching) - Its rule language makes ModSecurity an ideal external patching tool. External patching is all about reducing the window of opportunity. Time needed to patch application vulnerabilities often runs to weeks in many organizations. With ModSecurity, applications can be patched from the outside, without touching the application source code (and even without any access to it), making your systems secure until a proper patch is produced.
- Extrusion Detection Model (Outbound Proxy) - ModSecurity can also monitor outbound data and identify and block information disclosure issues such as leaking detailed error messages or Social Security Numbers or Credit Card Numbers.
Penetration Testing Tools:

- Firefox
  - Firebug
  - Live HTTP Headers
  - Tamper Data
  - Groundspeed
- OWASP
  - Web Scarab
- Application Scanners
  - Skipfish – Google Code
  - SQLMap
Learning Environments:

• Google Code University
  • Gruyere - cheesy web application (full of holes)
    • http://google-gruyere.appspot.com/
• Web Security
  • http://code.google.com/edu/security/index.html

• OWASP WebGoat – J2EE insecure Test App
  • https://www.owasp.org/index.php/Category:OWASP_WebGoat_Project

• McAfee
  • Hackme Bank
  • Hackme Travel
Free Books:
Malware-4dummies - free book from SANS:

Network Trouble Shooting with Wireshark: