Application Defense Tactics & Strategies
- WAF at the Gateway

Shreeraj Shah
Blueinfy Solutions Pvt. Ltd.

Dubai – HackInTheBox 2009

Who Am I?
- Founder & Director
  - Blueinfy Solutions Pvt. Ltd. (Brief)
  - SecurityExposure.com
- Past experience
  - Net Square, Chase, IBM & Foundstone
- Interest
  - Web security research
- Published research
  - Articles / Papers – Securityfocus, O’erilly, DevX, InformIT etc.
  - Tools – wsScanner, scanweb2.0, AppMap, AppCodeScan, AppPrint etc.
  - Advisories - .Net, Java servers etc.
- Books (Author)
  - Web 2.0 Security – Defending Ajax, RIA and SOA
  - Hacking Web Services
  - Web Hacking

Agenda
- Application Security Landscape
- Application Security Approaches
- Application Vulnerabilities - Demo
- WAF – A Quick Look
- .NET and HTTP processing
- Introducing IHttpModule
- Security Framework through set of Modules
- Conclusion

Methods – Concepts, Code Walk and Demos
Case of Portal

- Web 2.0 Portal – Buy / Sell
- Technologies & Components – Dojo, Ajax, XML Services, Blog, Widgets
- Scan with tools/products failed
- Security issues and hacks
  - SQL injection over XML
  - Ajax driven XSS
  - Several XSS with Blog component
  - Several information leaks through JSON fuzzing
  - CSRF on both XML and JS-Array

Case of Banking

- Scanning application for vulnerabilities
- Typical banking running with middleware
- Vulnerabilities
  - Profile manipulation (Logical and Hidden values)
  - XSS
  - Strong session management but URL rewriting
  - SQL is impossible in this case

Application Security State

- 95% companies hacked from web ports [FBI/CSI]
- 3 out of 4 web sites are vulnerable to attack (Gartner)
- Every 1500 lines of code has one security vulnerability (IBM Labs)
- 2000 attacks / week for unprotected web site
Next Generation Applications - 2.0

- 80% of companies are investing in Web Services as part of their Web 2.0 initiative (McKinsey 2007 Global Survey)
- By the end of 2007, 30 percent of large companies have some kind of Web 2.0-based business initiative up and running. (Gartner)
- 2008. Web Services or Service-Oriented Architecture (SOA) would surge ahead. (Gartner)

Real life Cases – 2.0

- Adding filter through CSRF
- Loading js file through flash from scrapbook
- Attacking blogs and boards
- XSS through RSS feed
- Flash components
- HTTP Response Splitting

Source: The Web Hacking Incidents Database [http://webappsec.org/projects/whid/]

Generic vectors 1.0/2.0

- Most common vulnerabilities

Source – Web Application Security Consortium

Generic threats – 1.0/2.0

- Threat types

Source – Web Application Security Consortium
**Root cause of Vulnerabilities**

CSI Security Survey : Vulnerability Distribution

- misconfiguration, other problems: 36%
- programming errors: 64%

**OWASP Top 10**

1. **Cross Site Scripting (XSS)**: XSS flaws occur whenever an application fails to validate or sanitize user supplied input before using it in a way that can be potentially harmful. XSS flaws allow attackers to execute scripts in the victim’s browser which can hijack user sessions, deface websites, redirect users to malicious sites, or plant malicious code on the victim’s device.

2. **Injection Flaws**: Injection vulnerabilities exist whenever an application constructs a query, command, URL, file path, or similar string using input from an untrusted source.

3. **Broken Authentication and Session Management**: These weaknesses are related to insecure ways in which data is transmitted between components, modules, programs, processes, threads, or systems.

4. **Sensitive Data Exposure**: This category includes vulnerabilities that expose sensitive data or allow attackers to conduct security-related attacks.

5. **Security Misconfiguration**: Misconfigured systems and services are often the largest cause of security incidents.

6. **XML External Entities (XXE)**: These vulnerabilities allow attackers to exploit flaws in how an application processes XML input, especially attack vectors initiated through XML APIs.

**CVE/CWE - Errors**

- **Insecure Interaction Between Components**
  - These weaknesses are related to insecure ways in which data is transmitted between components, modules, programs, processes, threads, or systems.
  - CWE-20: Improper Input Validation
  - CWE-21: Improper Encoding or Escaping of Output
  - CWE-22: Failure to Preserve SQL Query Structure (aka SQL Injection)
  - CWE-27: Failure to Preserve Web Page Structure (aka Cross-Site Scripting)
  - CWE-39: Improper Use of External Entities (aka XML External Entities)
  - CWE-41: Insecure Interaction Between Components
  - CWE-42: Improper Resource Shutdown or Release

- **Risky Resource Management**
  - These weaknesses are related to ways in which software does not properly manage the creation, usage, transfer, or destruction of important system resources.
  - CWE-119: Failure to Constrain Operations within the Bounds of a Memory Buffer
  - CWE-642: External Control of Critical State Data
  - CWE-73: External Control of File Name or Path
  - CWE-426: Untrusted Search Path
  - CWE-94: Failure to Control Generation of Code (aka Code Injection)
  - CWE-494: Download of Code Without Integrity Check
  - CWE-404: Improper Resource Shutdown or Release
  - CWE-665: Improper Initialization

Source - OWASP - http://owasp.org/

CVE/CWE - Errors

- **Porous Defenses**
  - The weaknesses in this category are related to defensive techniques that are often misused, abused, or just plain ignored:
  - CWE-327: Use of a Broken or Risky Cryptographic Algorithm
  - CWE-259: Hard-Coded Password
  - CWE-732: Insecure Permission Assignment for Critical Resource
  - CWE-330: Use of Insufficiently Random Values
  - CWE-250: Execution with Unnecessary Privileges
  - CWE-602: Client-Side Enforcement of Server-Side Security


PCI-DSS requirements

- To secure application
  - Put WAF at the gateway
  - Get your source code audited

New Attack Vectors

- XML manipulation
- SOAP and XML-RPC attacks and tempering
- CSRF with Ajax and Flash
- XSS with JSON streams
- Mashup and RSS attacks

Web Application Layout
### Attack Surface and Controls

- **Application Controls**: Accounts/Shares/Patches/Updates/Logging/Auditing/Port/Registry/OS/IPC/WU-FTPD/SUNRPC etc.
- **Web Services Business Application Level**: Application Level Web/Application Customization etc.
- **Services Level**: SSL/SMTP/POP etc.
- **Operating System Level**: Operating System Level Bandwidth/Buffering etc.
- **Firewall**: Added Defense

### Web 2.0 Architecture

- **Web 2.0 Start**: Documents, Weather, Bank/Trade, RSS feeds
- **Application Infrastructure**: Azure, AJAX, RIA (Flash), HTML/JS/DOM
- **Browser**: Internet, News, Mails, Blog

### Application Security Approaches

- **How to defend?**
  - **Two approaches**
    - Secure Coding and having proper validations at all levels to guard application layer. (Strategic)
    - Application layer traffic filtering to detect and block malicious requests/responses. (Tactical)
Secure Coding

- It is perfect and ideal approach.
- But...
  - Needs recoding
  - Takes longer time in fixing
  - Quick fix is required many times
  - QA process after changes
  - High cost
- Any work around?

Web Application Firewall (WAF)

- HTTP request and response filtering like traditional firewall.
- But it is specific to Application layer and rules should be well crafted.
- It is catching up and successful in detecting and blocking unintended traffic.
- It can block SQL injection, XSS, CSRF and many other attack vectors.

Application Vulnerabilities

- Let’s look at some vulnerabilities
  - SQL (JSON and Traditional)
  - XSS
  - XPATH
- Detecting it....
  - Scanning
  - Code Analysis

WAF – A Quick Look
Web Application Firewall (WAF)

- Advantages
  - Quick to add rules
  - Can act as first line of defense
  - No recoding is required
  - Easy to implement and manage
- Disadvantage
  - Performance a major hit
  - Rule based and bypass is possible

Application Infrastructure

SQL injection attack

http://store/products/display.asp?pg = 1&product = 7
SQL injection attack

http://store/products/display.asp?pg = 1 & product = 7

WAF models

- Following models are possible
  - Network traffic level filtering  
    [SSL is an issue]
  - Host level at Web Server
  - Host level + Reverse Proxy

.NET and HTTP processing

IIS architecture

- It is important to understand how IIS works?
- .NET gets integrated into IIS and applications can leverage the events
- IIS7.0 is coming up with a change that can help in building WAF
IIS higher level view

IIS 6.0 + ASP.NET

IIS 6.0 - Limitation

- ASP.NET is not having direct access to the HTTP pipe
- Can access ASP.NET requests only
- Framework is part of ISAPI and hooked to IIS
- Needs C++ based hooks to access generic pipe

Solved!

- IIS 7.0 – Change in Architecture
- Integrated mode
- .NET assemblies can be hooked directly to the pipe
- Full access to HTTP requests
- Can handle both .NET based as well as generic requests
- Access to all incoming requests…
IIS 7.0 – Integrated Mode

Introducing IHTTPModule

How to hook?
- Web application has separate scope and HTTP pipeline can be accessed.
- HTTP request can be accessed before it hits application resources.
- HTTPModule and HTTPHandler are defense at your gates. …

HTTP pipe for .NET
**Interfaces and Hooks**

- HttpRuntime
- HttpApplicationFactory
- HttpApplication
- IHttpModule
- HttpHandlerFactory
- Handler

**Leveraging Interfaces**

- HTTPModule and HTTPHandler - can be leveraged.
- Application layer firewall can be cooked up for your application.
- Similarly IDS for web application can be developed.
- It sits in HTTP pipe and defend web applications.

**For IIS 7.0**

- Integrated mode with full access
- Possible to cook up reverse proxy as well
- Traffic can be controlled at the gates
- Sound defense can be created with minimal coding
- Your module can be on top of the pipe
- Can access
  - HttpResponse.Headers
  - HttpRequest.Headers
  - HttpRequest.ServerVariables

**Implementing IHttpModule**
**IHTTPModule**

- Managed code in C# can be hooked into HTTP pipe.
- Module can help in filtering HTTP requests.
- Let’s see its implementation.

```csharp
public class iAppFilter : IHttpModule {
}
```

**HttpApplication**

**Event Mapping**
Event Trapping and Firewall

Accessing HTTP request
- Access with BeginRequest
- Access to Http Context
- Access to headers
- All server variable
- Complete access for filtering

Hooking to HTTP pipe

```csharp
public void Init(HttpApplication application)
{
    application.BeginRequest +=
        (new EventHandler(this.Application_BeginRequest));
}

private void Application_BeginRequest(Object source,
    EventArgs e)
{
    HttpApplication application = (HttpApplication)source;
    HttpContext context = application.Context;
}
```

Processing POST

```csharp
if (app.Request.ServerVariables["REQUEST_METHOD"] == "POST")
{
    long streamLength = app.Request.InputStream.Length;
    byte[] contentBytes = new byte[(int)streamLength];
    app.Request.InputStream.Read(contentBytes, 0, (int)streamLength);
    postreq = System.Text.Encoding.UTF8.GetString(contentBytes);
}
```
Request / Response

Putting it in action
- DLL get created after compilation
- Module in Bin folder
- Adding to config file
- It is different with IIS 7.0 for integrated mode
- Directives are different
- Let’s see in detail

Security Framework through set of Modules

Accessing all requests
- It is important to access all incoming HTTP requests
- IIS 6.0 limitation – can be overcome by using wildcard mapping [Some what]
- IIS 7.0 – Leveraging integrated mode
IIS 6.0 – Wildcard mapping

IIS 7.0 – Integrated mode

```
<modules>
  <add name="iAppWall" type="iAppWall"/>
</modules>
```

Security Modules

- Various module can be cooked up.
- Authorization, Authentication, Filtering, XML processing, IDS etc.
- All of them can be part of one DLL or multiple.

Authorization Module

- Limited access to IP addresses
- Blocking sensitive directories
- Session based access to various area of application
Validation Module
- Detecting attack vectors like XSS or SQL injection
- Blocking those requests at the module level
- Total security to all incoming parameters both over GET and POST

Web 2.0 Security Module
- Web 2.0 runs on XML, JSON, JS-Array etc..
- Intelligent module to detect these sort of traffic and block malicious requests
- Protecting Web Services running over SOAP, XML/JSON-RPC, REST etc.

CSRF Defense Module
- Cross Site Request Forgery is a big concern for sensitive forms
- Protection by referrer tag or token by HTTP module
- Securing application against CSRF attack vectors

Response Filtering Module
- Limited response filtering for critical resources
- Monitoring outgoing requests
- Capturing suspicious traffic and blocking them
- Web 2.0 framework defense – RSS or proxy based responses
**IDS Module**

- Logging all suspicious requests for forensic use
- Logging and monitoring can be improved
- Logging to central database, file or OS events.

**Reverse Proxy Module**

- Defending non IIS applications with reverse tunneling.
- IIS 7.0 as front end server and securing internal servers
- Complete control over full traffic going in/out

**Conclusion**

- Next generation .NET application can be defended by IHttpModules
- IIS 7.0 – Integrated mode is going to play a big role
- Web 2.0 application needs better filtering capabilities and IHttpModule can deliver it

**Questions**

http://shreeraj.blogspot.com
shreeraj@blueinfy.com
http://www.blueinfy.com