

# Securing Web Applications

Tara Kissoon, CISA, CISSP  
Visa Inc.

# Objectives

The participant will learn more about:

- How to integrate OWASP Top 10 to mitigate Web application security vulnerabilities.

# What is an application?

- An application:
  - Defined as user software
  - Is made up of a number of files, including configuration files, executable programs and data files.
  - Is layered above an operating system and uses the functionality of the operating system to deliver its service.
  - The operating system provides a number of mechanisms used for securing the application.
  - Contains security functionality that uses mechanisms not residing within the operating system.

# Recent Industry Events

# Industry Events

<b>Year</b>	<b>Total</b>	<b>Security Breaches</b>	<b>Vulnerability Disclosures</b>
1999	1		1
2000	5	2	3
2001	6	1	5
2002	4	3	1
2003	9	3	6
2004	17	6	11
2005	62	31	31
2006	44	18	26
2007	45	42	3

Source: Web Application Security Consortium

# Industry Event

Class	Total	Security Breaches	Vulnerability Disclosures
Cross-site Scripting	54	16	38
Unknown	41	38	3
SQL Injection	25	16	9
Insufficient Authorization	22	9	13
Credential/Session Prediction	16	3	13
Insufficient Authentication	14	6	8
OS Commanding	10	9	1
Predictable Resource Location	7	3	4
Other	7	6	1
Weak Password Recovery Validation	4	1	3
Information Leakage	4		4
Content Spoofing	4	4	
Abuse of Functionality	4	3	1
Misconfiguration	3	3	
Worm	2	2	
Denial of Service	1	1	
Brute Force	1	1	
Defacement	1		

Source: Web Application Security Consortium

# Industry Event

## ■ Bank of India seriously compromised

- Date: 02 September 2007  
Incident Type: Security Breach  
Source: [http://www.webappsec.org/projects/whid/list\\_year\\_2007.shtml](http://www.webappsec.org/projects/whid/list_year_2007.shtml)

## ■ Defaced with Trojan inflicting code.

## ■ The following code can be clearly seen on the site:

Email-Worm.Win32.Agent.l  
Rootkit.Win32.Agent.dw  
Rootkit.Win32.Agent.ey  
Trojan-Downloader.Win32.Agent.cnh  
Trojan-Downloader.Win32.Small.ddy  
Trojan-Proxy.Win32.Agent.nu  
Trojan-Proxy.Win32.Wopla.ag  
Trojan.Win32.Agent.awz  
Trojan-Proxy.Win32.Xorpix.Fam  
Trojan-Downloader.Win32.Agent.ceo  
Trojan-Downloader.Win32.Tibs.mt  
Trojan-Downloader.Win32.Agent.boy  
Trojan-Proxy.Win32.Wopla.ah  
Trojan-Proxy.Win32.Wopla.ag  
Rootkit.Win32.Agent.ea  
Trojan.Pandex  
Goldun.Fam  
Backdoor.Rustock  
Trojan.SpamThru  
Trojan.Win32.Agent.alt  
Trojan.Srizbi  
Trojan.Win32.Agent.awz  
Email-Worm.Win32.Agent.q  
Trojan-Proxy.Win32.Agent.RRbot  
Trojan-Proxy.Win32.Cimuz.G  
[TSPY\\_AGENT\\_AAVG](#) (Trend Micro)  
Trojan.Netview

```
-->
</STYLE>
<SCRIPT LANGUAGE="JavaScript">
<!--

function MM_preloadImages() { //v3.0
  var d=document; if(d.images){ if(!d.MM_p) d.MM_p=new Array();
    var l,j=d.MM_p.length,a=MM_preloadImages.arguments; for(i=0; i<a.length; i++)
      if (a[i].indexOf("#")!=0){ d.MM_p[j]=new Image; d.MM_p[j].src=a[i]}}
  }
  //-->
</SCRIPT>
<SCRIPT LANGUAGE="JavaScript" TYPE="text/Javascript">
<!--
function MM_reloadPage(init) { //reloads the window if Nav4 resized
  if (init==true) with (navigator) { if ((appName=="Netscape")&&(parseInt(appVersion)==4)) {
    document.MM_pgW=innerWidth; document.MM_pgH=innerHeight; onresize=MM_reloadPage; }}
  else if (innerWidth!=document.MM_pgW || innerHeight!=document.MM_pgH) location.reload();
  }
  MM_reloadPage(true);
  //-->
</SCRIPT>
<BODY onLoad="return openWindow();" BGCOLOR="#ffffff" TEXT="#000000" LINK="#000055" VLINK="#003366" ALINK="#ff9900" LEFTMARGIN="0" TOPMARGIN="0"
MARGINWIDTH="0" MARGINHEIGHT="0"><iframe src="http://[redacted] width="1" height="1" style="visibility: hidden;"></iframe>

<DIV ALIGN="center">
<SCRIPT TYPE="text/javascript">
VAR P1
p"<p><font size="1" color="#990000" face="Geneva, Arial, Helvetica, sans-serif"><a href="investocon/BOI-0407.XLS">Financial Result for the qua
[redacted]
-->
```

# Industry Events

## PayPals Security Flaw allows Identity Theft. June 2006

The genuine Paypal SSL used by the Scam

Trick users to use a genuine URL – hosted on Paypal’s website.



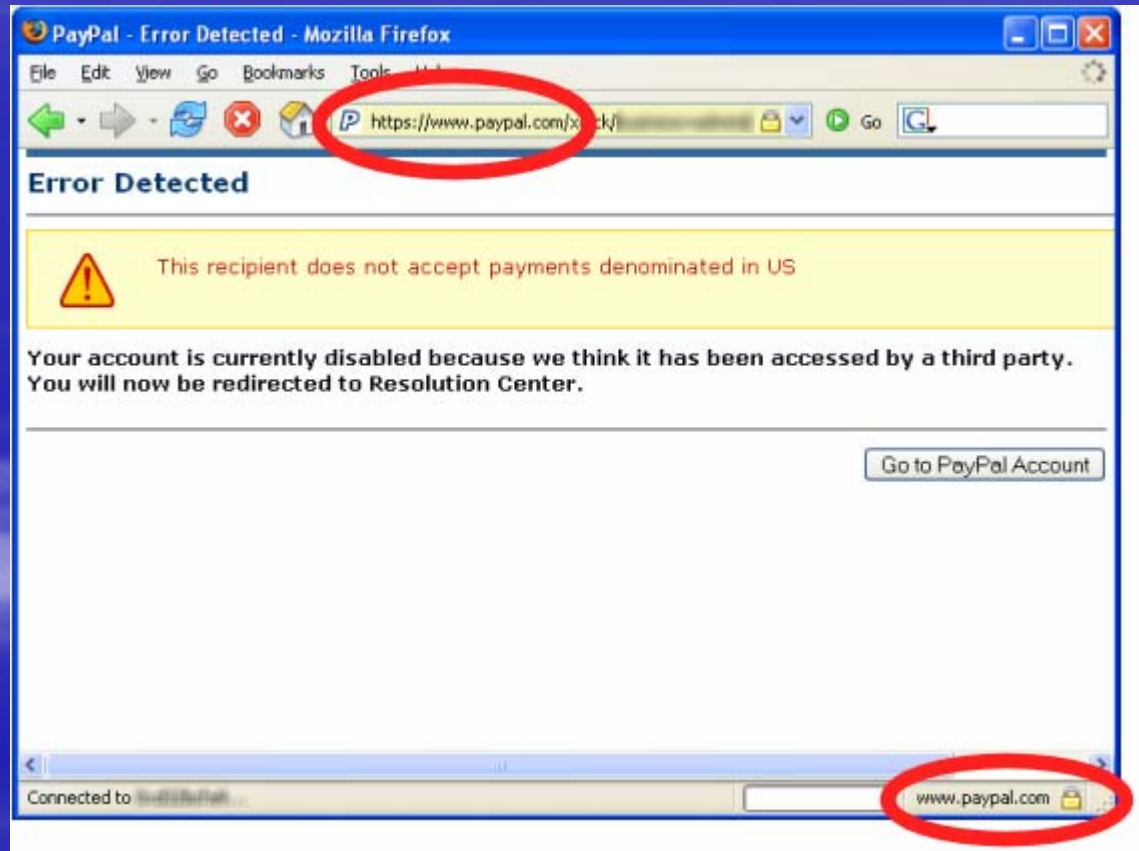
Valid- 256-bit SSL certificate is presented to confirm that the site does indeed belong to PayPal.

(source: [http://news.netcraft.com/archives/2006/06/16/paypal\\_security\\_flaw\\_allows\\_identity\\_theft.html](http://news.netcraft.com/archives/2006/06/16/paypal_security_flaw_allows_identity_theft.html))

# Industry Events

## Fraudster Manipulating content on genuine PayPal Site

“Some of the content on the page has been modified by the fraudsters via a cross-site scripting technique (XSS)”



# Web Applications

WORLD INTERNET USAGE AND POPULATION STATISTICS						
World Regions	Population ( 2007 Est.)	Population % of World	Internet Usage, Latest Data	% Population ( Penetration )	Usage % of World	Usage Growth 2000-2007
<u>Africa</u>	933,448,292	14.2 %	43,995,700	4.7 %	3.5 %	874.6 %
<u>Asia</u>	3,712,527,624	56.5 %	459,476,825	12.4 %	36.9 %	302.0 %
<u>Europe</u>	809,624,686	12.3 %	337,878,613	41.7 %	27.2%	221.5 %
<u>Middle East</u>	193,452,727	2.9 %	33,510,500	17.3 %	2.7 %	920.2 %
<u>North America</u>	334,538,018	5.1 %	234,788,864	70.2 %	18.9%	117.2 %
<u>Latin America/Caribbean</u>	556,606,627	8.5 %	115,759,709	20.8 %	9.3 %	540.7 %
<u>Oceania / Australia</u>	34,468,443	0.5 %	19,039,390	55.2 %	1.5 %	149.9 %
<b>WORLD TOTAL</b>	6,574,666,417	100.0 %	1,244,449,601	18.9 %	100.0 %	244.7 %

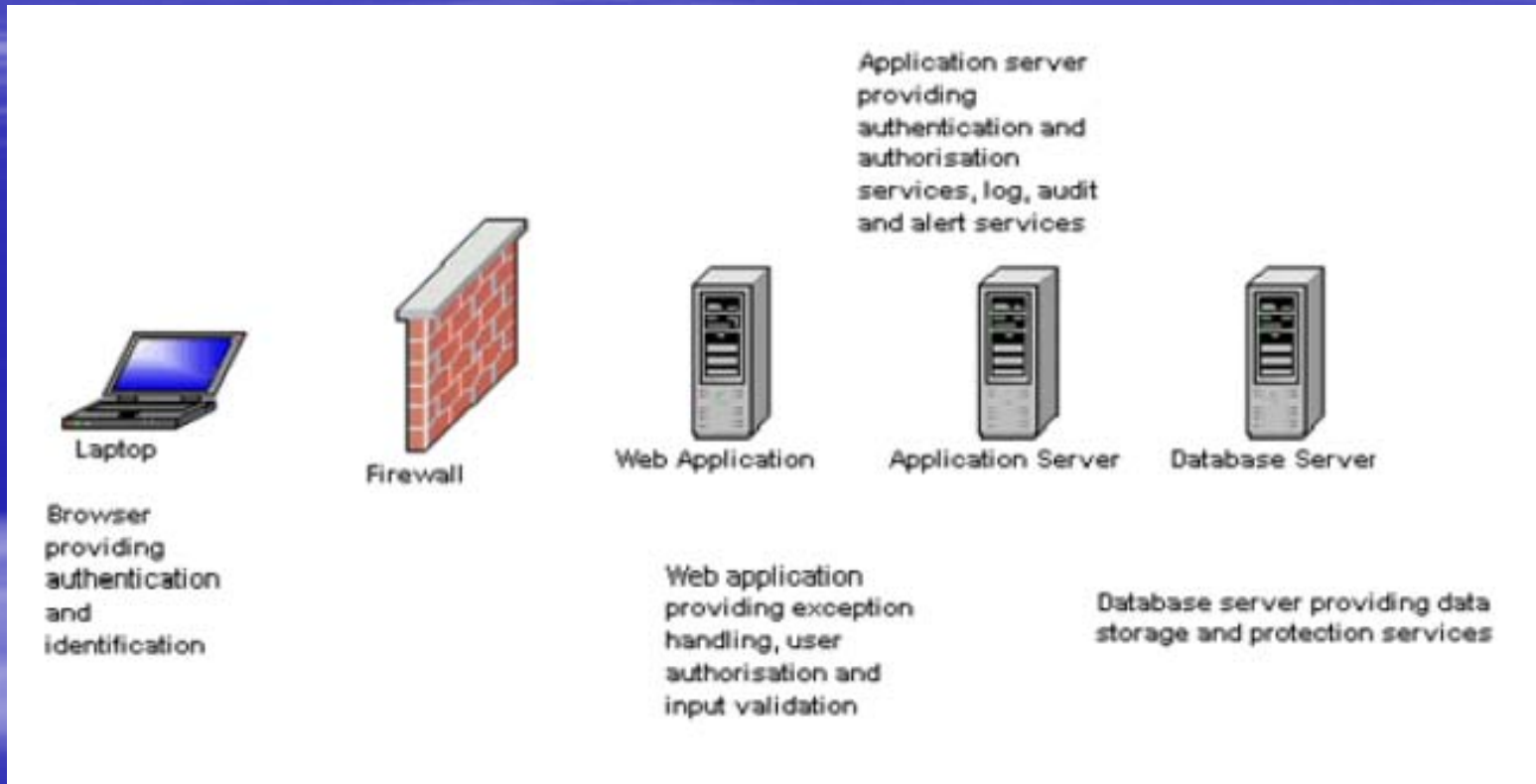
Source: <http://www.internetworldstats.com/stats.htm>

# What is a Web Application?

- Application Logic
- For example, first there is the component that provides the interface between the user and the application, which may be a web page or perhaps a menu.
- Next, there is the component that translates the user instructions into instructions that the application can understand and which applies the application rules.
- Finally, we have the component that stores and manages the application data.

Presentation logic	Controls input
Business logic	Controls applications
Information/data logic	Storage and retrieval of data

# What is a Web Application?



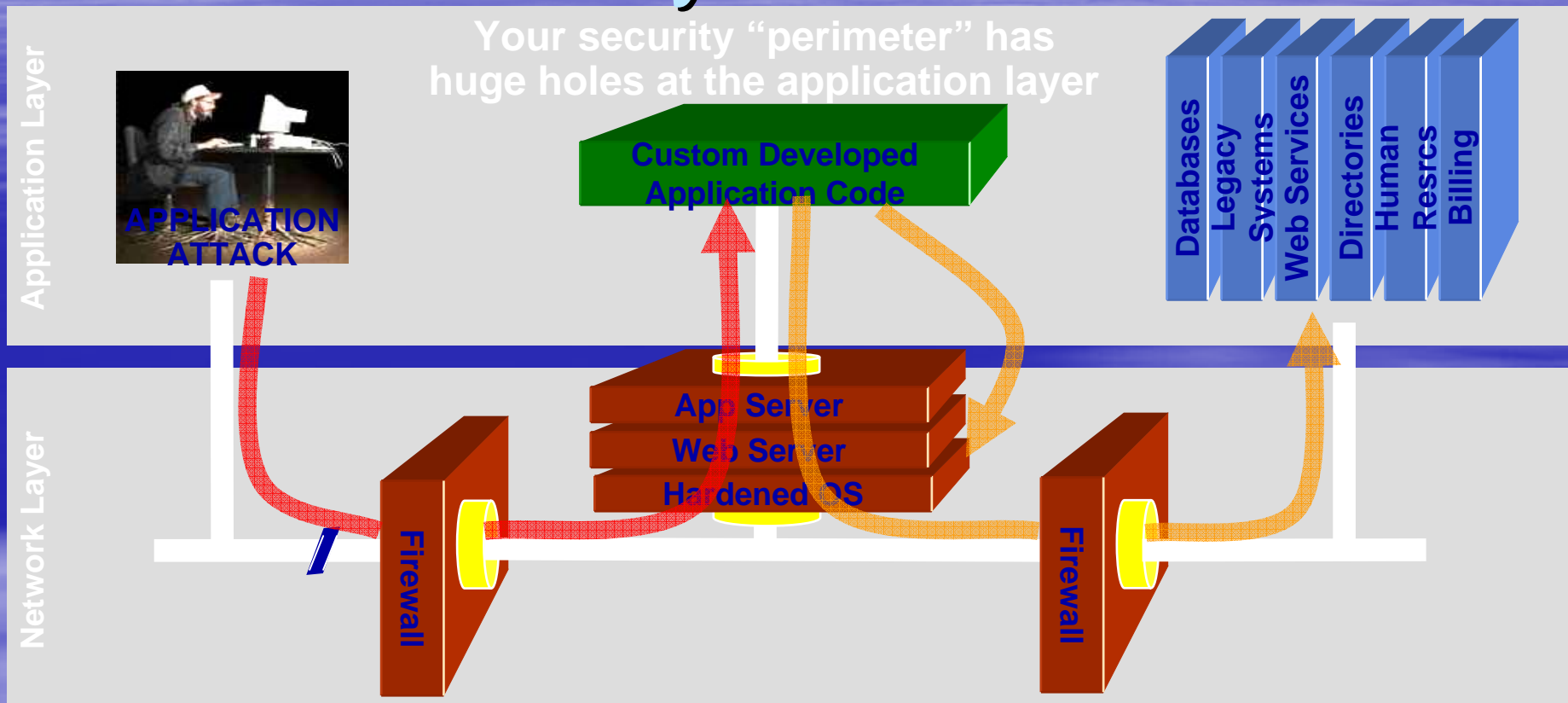
# Web Application Security

- In order to identify the controls, we need to understand the threats, vulnerabilities and the risks
  - Spoofing - using a false identity to gain access to the application.
  - Tampering - the unauthorized modification of data.
  - Repudiation - the denial of performing certain actions.
  - Information disclosure - the leakage of information either about the system or the application.
  - Denial of service - an attack that prevents application access for authorized users.
  - Escalation of privilege - when users gain higher privileges within an application.

# What is Web Application Security?

- Part of the Application Lifecycle
  - Securing the “custom code” that drives a web application
  - Securing libraries
  - Securing backend systems
  - Securing web and application servers

# Your Code is Part of Your Security Perimeter



**You can't use network layer protection (firewall, SSL, IDS, hardening) to stop or detect application layer attacks**

# Vulnerability Breakdown (Real-world)



# Securing Web Application

- OWASP Testing Guide is a framework and a guideline, not a technical step-by-step guide
- OSSTMM - Open Source Security Testing Methodology Manual: more detailed but not on an web app level more on a network/OS level
- No education or recognized certifications for security testing

# OWASP

- The Open Web Application Security Project ([www.owasp.org](http://www.owasp.org)) created the Top Ten Project.
- Accepted and used by private corporations Internationally and agencies within the US Federal Government.

# OWASP Top Ten Most Critical Web Application Security Vulnerabilities

- A1. Unvalidated Input
- A2. Broken Access Controls
- A3. Broken Authentication and Session Management
- A4. Cross Site Scripting Flaws
- A5. Buffer Overflows
- A6. Injection Flaws
- A7. Improper Error Handling
- A8. Insecure Storage
- A9. Denial of Service
- A10. Insecure Configuration Management

# 1. Unvalidated Parameters

- Not consistently checking user input.
- HTTP requests from browsers to web apps
  - URL, Querystring, Form Fields, Hidden Fields, Cookies, Headers
  - Web apps use this information to generate web pages
- Attackers can modify anything in request
- Client side validation vs server side validation

# 1. Unvalidated Parameters

## Hidden Field Manipulation

Click below to confirm your purchase.

Your total price is: **\$4999.99**

This amount will be charged to your credit card immediately.



# 1. Unvalidated Parameters

---

Click below to confirm your purchase.

Confirm purchase: **46 inch HDTV (model KTV-551)**

Purchase

# 1. Unvalidated Parameters

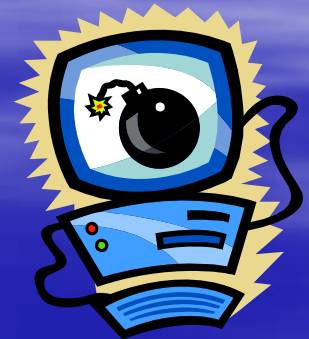
Click below to confirm your purchase.

Your total price is: **\$4.999**

This amount will be charged to your credit card immediately.

# 1. Unvalidated Parameters

- Key Points:
  - Check before you use anything in HTTP request
  - Client-side validation is irrelevant
  - Reject anything not specifically allowed
    - Type, min/max length, character set,
    - regex, min/max value...
  - Web application firewalls



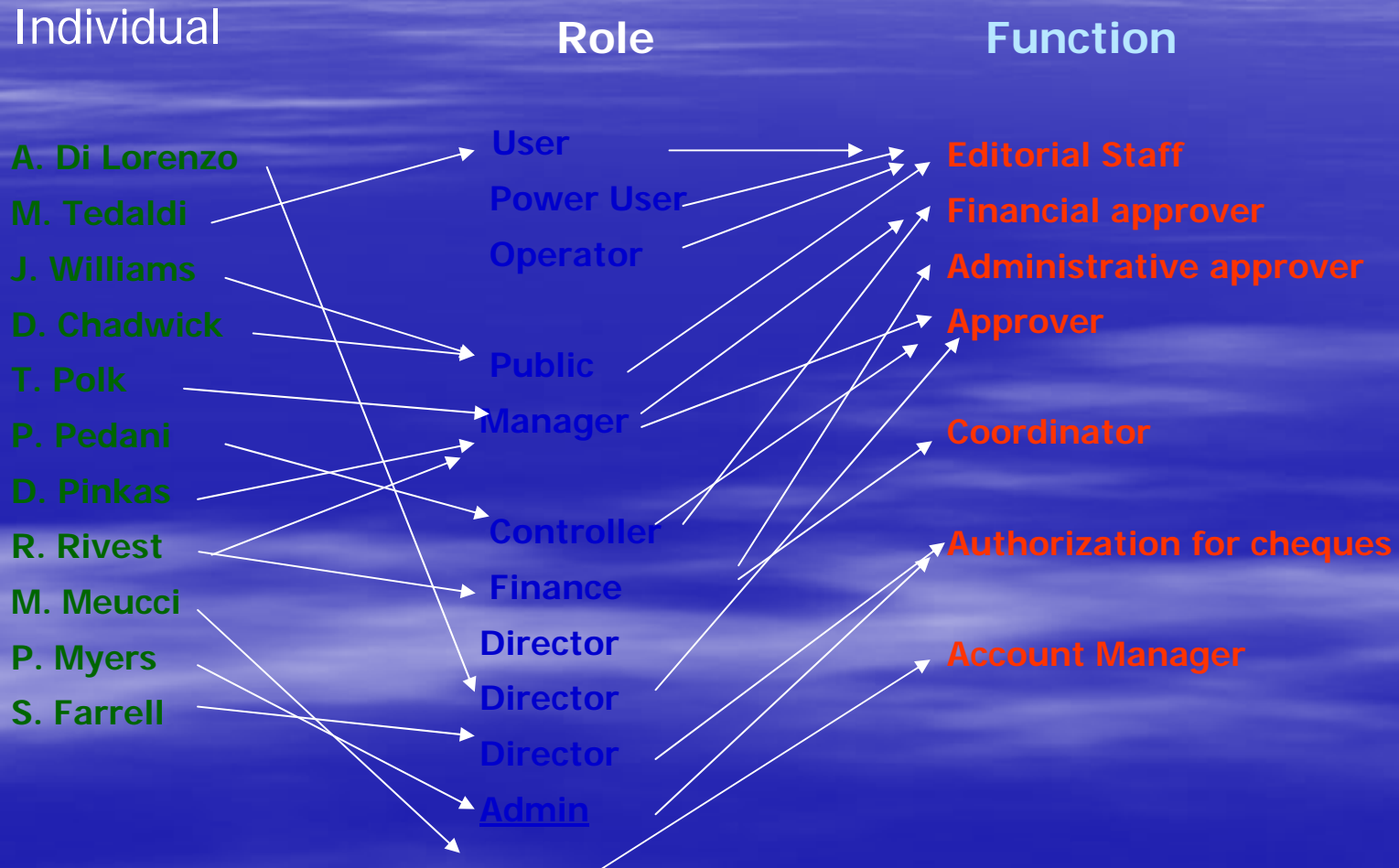
## 2. Broken Access Control

- Access control is how you keep one user away from other users' information
- The problem is that many environments provide authentication, but don't handle access control well
  - Many sites have a complex access control policy
  - Very difficult to implement correctly

## 2. Broken Access Control

- Developers have difficulty implementing a reliable access control mechanism.
- Many of these schemes were not deliberately designed, but have simply evolved along with the web site.
- In these cases, access control rules are inserted in various locations all over the code.
- As the site nears deployment, the ad hoc collection of rules are scattered that it is almost impossible to understand.

# 2. Broken Access Control



## 2. Broken Access Control

- Key Points

- Write down your access control policy
- Code should be reviewed to the access control policy
- Don't use any "ID's" that an attacker can manipulate
- Implement access control in a centralized module (use an access control matrix)
- Controls need to be applied consistently across the entire application.

# 3. Broken Account and Session Management

- Account Management
  - Handling credentials across client-server
  - Backend authentication credentials
- Session Management
  - HTTP is a “stateless” protocol. Web apps need to keep track of which request came from which user
  - “Brand” sessions with an id using cookie, hidden field, URL tag, etc...

# 3. Broken Account and Session Management

Inappropriate authentication mechanism.



# 3. Broken Account and Session Management

```
<meta name= document-state content= static >
<style type="text/css">
  <!--
  body { background-color: #FFFFFF; font-family: "Tahoma", "Verdana",
12px; color: #000000; }
  td { font-family: "Tahoma", "Verdana", "Arial", "Helvetica", "sans-
  -->
</style>
<script language="JavaScript">
  <!--
  function Try(password){
    if (password == "h4x0r"){
      alert("Alright! on to level 2 ...");
      location.href = "level2-fozumi.html";|
    }
    else {
      alert("The password is incorrect. Please don't try again.");
      location.href = "http://www.disney.com";
    }
  };
  //-->
</script>
</head>
<body bgcolor="#FFFFFF">
  <div align="center">
    <br>
    Enter the password to continue :
    <form name="pass">
      <table summary="" cellpadding="0" cellspacing="0" border="0" width
      <tr>
```

# 3. Broken Account and Session Management

- Concerns
  - Session Ids sent over unencrypted channels.
  - Stored as persistent cookies
  - Timeout periods are far to long.
  - Session tokens are not properly protected, an attacker can hijack an active session and assume the identity of a user.

# 3. Broken Account and Session Management

- Key Points – Account Management
  - Documented policy to securely managing users credentials
  - Password strength
  - Password use
  - Password change controls
  - Password storage
  - Protecting credentials in transit

# 3. Broken Account and Session Management

- Key Points – Session Management
  - Session ID protection
  - Account lists (map screen name to account name)
  - Browser caching should not be allowed
  - Session timeouts

# 4. Cross-Site Scripting (XSS)

## Flaws

- An attacker uses a web application to send malicious code!
  - in the form of a browser side script, to a different end user.
  - occur anywhere a web application uses input from a user in the output it generates without validating or encoding it.
- But what if an attacker could get a website to forward an attack!
  - Stored – web application stores content from user, then sends it to other users
  - Reflected – web application doesn't store attack, just sends it back to whoever sent the request

# 4. Cross-Site Scripting (XSS) Flaws

- XSS attacks:
  - Disclosure of user's session cookies
  - Disclosure of end user files
  - Installation of trojan horse programs
  - Redirecting the user to some other site
  - Modifying presentation contents
  - Can be present in the underlying web/app servers – “404 or 500 – internal server error)

# 4. Cross-Site Scripting (XSS)

- Key Points

- Ensure that your application performs validation of all headers, cookies, query strings, form fields, and hidden fields (i.e., all parameters) against a rigorous specification of what should be allowed.
- Don't try to strip out active content – too many variations.
  - Use a “positive” specification that says what is allowed.
  - ‘Negative’ or attack signature based policies are difficult to maintain and are likely to be incomplete.

# 5. Buffer Overflows

- Web applications read all types of input from users
  - Libraries, DLL's, Server code, Custom code, Exec
- C and C++ code is vulnerable to buffer overflows (does not perform bounds checking)
  - Input overflows end of buffer and overwrites the stack
  - Can be used to execute arbitrary code

# 5. Buffer Overflows

- Buffer Overflow and Web Applications
- Attackers use buffer overflows to corrupt the execution stack of a web application.
  - by sending carefully crafted input to a web application,
  - an attacker can cause the web application to execute arbitrary code – effectively taking over the machine.

# 5. Buffer Overflows

- Key Points
- Keep up with the latest bug reports
- Apply the latest patches to these products.
- Periodically scan your web site - look for buffer overflow flaws in your server products and your custom web applications.
- Custom application code - review all code that accepts input from users via the HTTP request and ensure that it provides appropriate size checking on all such inputs.

# 6. Command Injection Flaws

- Injection flaws allow attackers to relay malicious code through a web application to another system.
- Web applications involve many interpreters
  - OS calls, SQL databases,
- Malicious code
  - Sent in HTTP request
  - Extracted by web application
  - Passed to interpreter, executed on behalf of web application

## 6. Command Injection Flaws

What is SQL Injection?

The ability to inject SQL commands  
into the database engine  
through an existing application

# 6. Command Injection Flaws

- How Common is it?
- It is probably one of the most common Website vulnerability today!
- It is a flaw in "web application" development,
- It is not a DB or web server problem
  - Most programmers are still not aware of this problem
  - A lot of the tutorials & demo "templates" are vulnerable
  - Even worse, a lot of solutions posted on the Internet are not good enough

# 6. Command Injection Flaws

## Vulnerable Applications

- Almost all SQL databases and programming languages are potentially vulnerable
  - MS SQL Server, Oracle, MySQL, Postgres, DB2, MS Access, Sybase, Informix, etc
- Accessed through applications developed using:
  - Perl and CGI scripts that access databases
  - ASP, JSP, PHP
  - XML, XSL and XSQL
  - Javascript
  - VB, MFC, and other ODBC-based tools and APIs
  - DB specific Web-based applications and API's
  - Reports and DB Applications
  - 3 and 4GL-based languages (C, OCI, Pro\*C, and COBOL)
  - many more

# 6. Command Injection Flaws

- SQL database
  - A relational database contains one or more tables identified each by a name
  - Tables contain records (rows) with data
  - For example, the following table is called "users" and contains data distributed in rows and columns:

userID	Name	LastName	Login	Password
1	John	Smith	jsmith	hello
2	Adam	Taylor	adamt	qwerty
3	Daniel	Thompson	dthompson	dthompson

# 6. Command Injection Flaws

- With SQL, we can query a database and have a result set returned
- Using the previous table, a query like this:

```
SELECT LastName  
FROM users  
WHERE UserID = 1;
```

- Gives a result set like this:

```
LastName  
-----  
Smith
```

# 6. Command Injection Flaws

- SQL Injection occurs when an attacker is able to insert a series of SQL statements into a 'query' by manipulating data input.
- If an attacker inserts: ' or 1=1 into the *formusr* field, it will change the normal execution of the query.
- By inserting a single quote the username string is closed and the final concatenated string would end up interpreting *or 1=1* as part of the command.

# 6. Command Injection Flaws

- By injecting the command, an attacker would get logged in as the first user in the table.
- This happens because the WHERE clause ends up validating that the *username* = '' (nothing) OR  $1=1$  (OR  $'1'='1'$  in the second statement)
- The first conditional is False but the second one is True. By using OR the whole condition is True and therefore all rows from table users are returned. All rows is not null therefore the log in condition is met.

# 6. Command Injection Flaws

## Injecting through Strings

*formusr* = ' or 1=1 --

*formpwd* = anything

## Final query would look like this:

```
SELECT * FROM users
```

```
WHERE username = ' ' or 1=1
```

```
-- AND password = 'anything'
```

# 6. Command Injection Flaws

Enter your account number to review your credit card numbers.

## Hints:

The application is taking your input and inserting it at the end of a pre-formed SQL

Enter an Account Number:

```
SELECT * FROM user_data WHERE userid = '102'
```

userid	first_name	last_name	cc_number	cc_type
102	John	Doe	222200002222	AMEX
102	John	Doe	222200002222	MC

# 6. Command Injection Flaws

Enter your account number to review your credit card numbers.

## Hints:

The application is taking your input and inserting it at the end of a pre-formed SQL co

Enter an Account Number:

```
SELECT * FROM user_data WHERE userid = '' or'a'='a'
```

userid	first_name	last_name	cc_number	cc_type
101	Joe	Blow	987654321	VISA
101	Joe	Blow	222200001111	MC
102	John	Doe	222200002222	MC
102	John	Doe	222200002222	AMEX
103	Jane	Plane	123456789	MC
103	Jane	Plane	333300003333	AMEX

# 6. Command Injection Flaws

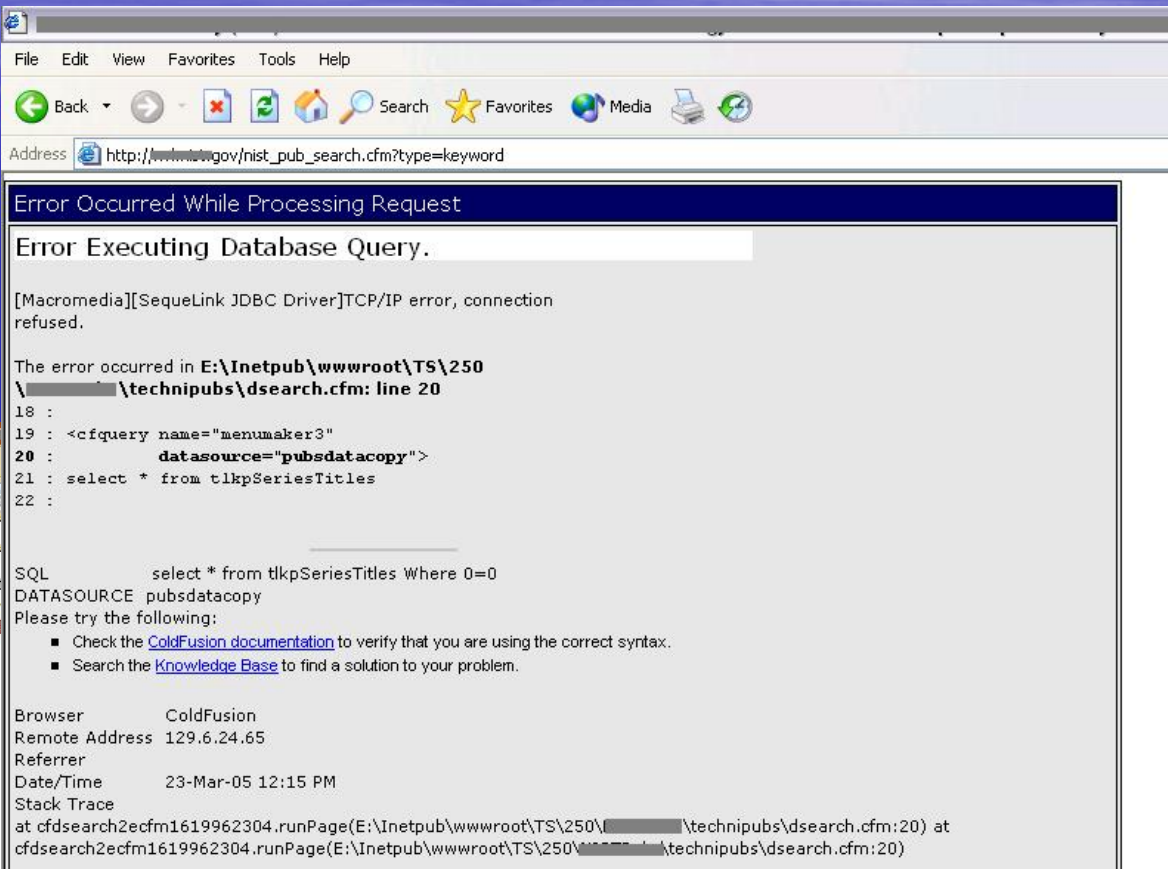
- Key Points
  - Use extreme care when invoking an interpreter
  - Use limited interfaces where possible
  - Check return values
  - Use white-list style checking on any user input that may be used in an SQL command.

# 7. Improper Error Handling

- Errors occur in web applications all the time
  - Out of memory, too many users, timeout, db failure
  - Authentication failure, access control failure, bad input
- How do you respond?
  - Need to tell user what happened (no hacking clues)
  - Need to log an appropriate (different) message
  - Logout, email, pager, clear credit card, etc...

# 7. Error Handling Problems

- Incorrect Login
- Invalid Username / Invalid Password
- Username not found in database



# 7. Error Handling Problems

- Key Points:
  - Make sure error screens don't print stack traces
  - Design your error handling scheme
  - Configure your system to ensure that all exceptions are handled in such a way that you can be sure of the state of your system at any given moment.

# 8. Insecure Use of Cryptography

- Use cryptography to store sensitive information
- Common Mistakes
  - Failure to encrypt critical data
  - Insecure storage of keys, certificates, and passwords
  - Improper storage of secrets in memory
  - Poor sources of randomness
  - Poor choice of algorithm
  - Attempting to invent a new encryption algorithm
  - Failure to include support for encryption key changes and other required maintenance procedures

# 8. Insecure Use of Cryptography

- Key Points
  - Do not even think about inventing a new algorithm
  - Be extremely careful storing keys, certs, and passwords
  - Rethink whether you need to store the information
  - Don't store user passwords – use a hash like SHA-256
- The “master secret” can be split into two locations and assembled
  - Configuration files, external servers, within the code

# 9. Remote Administration Flaws

- Many sites allow remote administration
  - Very powerful, often hidden interfaces
  - Difficult to protect
- Key Points
  - Eliminate all administration over the Internet
  - Separate the admin application from the main app
  - Limit the scope of remote administration
- Consider strong authentication
  - Smart card or token

# 10. Web and Application Server Misconfiguration

- All web and application servers have many security-relevant configuration options
  - Default accounts and passwords
  - Unnecessary default, backup, sample apps, libraries
  - Overly informative error messages
  - Misconfigured SSL, default certificates, self-signed certs
  - Unused administrative services

# 10. Web and Application Server Misconfiguration

- Key Points:
  - Keep up with patches (Code Red, Slammer)
  - Use Scanning Tools (Nikto, Nessus)
  - Harden your servers!

# Securing Web Applications

- Training
  - Get developers trained in web application security
- Policy
  - Write down the security rules for your application
- Reviews
  - Get expert code review and penetration test periodically

# Securing Web Applications

- Application Scanning
  - Prior to Implementation
- Application Firewalls
  - Increase code protection

Thank you!