



JavaOne

You Are Hacked ☹️: AJAX Security Essentials for Enterprise Java™ Technology Developers

Karthik Shyamsunder

Principal Engineer
VeriSign, Inc.

James Gould

Principal Engineer
VeriSign, Inc.

Session TS-6014

Speaker Qualifications

- Karthik Shyamsunder
 - Principal Engineer, VeriSign, Inc.
 - Adjunct Faculty at Johns Hopkins University
- James Gould
 - Principal Engineer, VeriSign, Inc.
 - Application Architect for VeriSign Naming Services

Overall Presentation Goal

What will you learn?

1. Understand the AJAX security model
2. Identify the various threats to your AJAX web applications
3. Learn what you can do to protect your application from these threats



Agenda

- Internet Threat Model
- Browser Security Model
- Vulnerabilities, Attacks, & Countermeasures
- Secure Software-Development Process
- Summary
- Q&A



Agenda

- **Internet Threat Model**
- Browser Security Model
- Vulnerabilities, Attacks, & Countermeasures
- Secure Software Development Process
- Summary
- Q&A

JavaScript worm targets Yahoo!



Malware latches onto unpatched flaw

By [John Leyden](#) → [More by this author](#)

Published Monday 12th June 2006 15:28 GMT

Research library - All papers free to download.

A JavaScript worm that takes advantage of an unpatched vulnerability in Yahoo!'s webmail service has been discovered on the net.

The JS-Yamanner worm spreads when a Windows user accesses Yahoo! Mail to open an email sent by the worm. The attack works

Cross-Site Scripting Worm Hits MySpace

By [Nate Mook](#), BetaNews
October 13, 2005, 6:28 PM

With the advent of social [networking](#) sites, becoming more popular, a few lines of JavaScript code, it seems.

One clever [MySpace](#) user looking to expand his buddy list recently found a way to trick others into becoming his friend, and ended up creating the first self-p

Netflix fixes Web 2.0 bugs

By [Joris Evers](#), CNET News.com

Published on [ZDNet News](#): October 16, 2006, 5:05 PM PT



TALKBACK
ADD YOUR OPINION

Worthwhile?



+

0 VC

ZDNet Tags: [Entertainment](#), [Security](#),

Netflix has fixed weaknesses in its Web site that could have let outsiders change a user's address, add movies to their rental queue, and potentially hijack their account.



■ Home ■ News ■ Travel ■ Money ■ Sports ■ Life ■

Money ▾

■ GET A QUOTE: GO ■ DJIA 12,161.92 ▲ +28.52 ■ NASDAQ 2,378.

Cybercrooks add Ajax coding to bag of hacking tricks

Updated 8/4/2006 3:33 AM ET

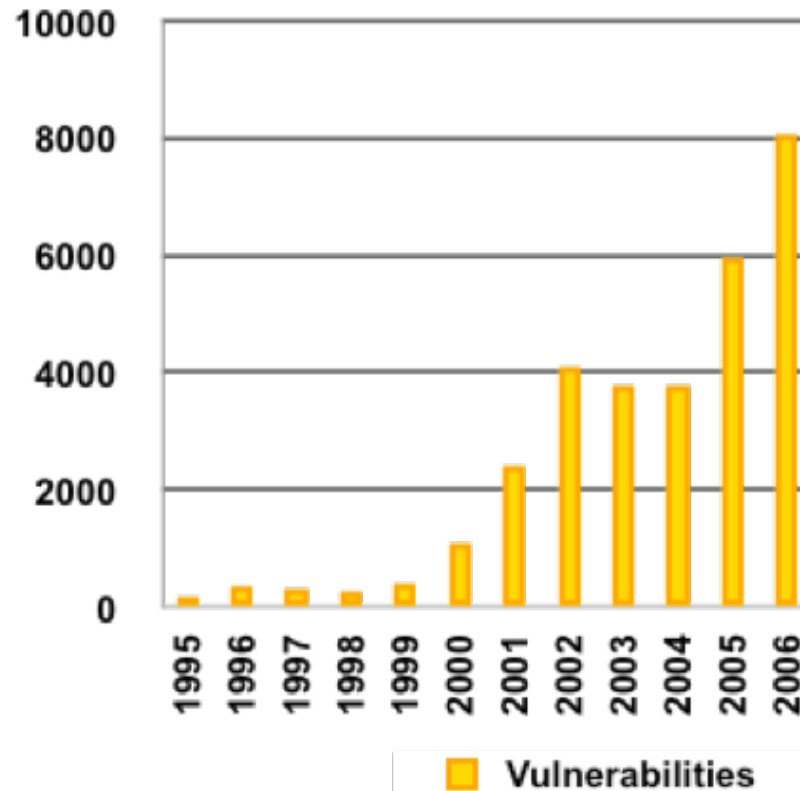
[E-mail](#) | [Save](#) | [Print](#) | [Reprints & Permissi](#)

By [Byron Acolhido](#) and [Jon Swartz](#), USA TODAY



The Problem Is Real

- Cyber crimes and incidents are on the rise
- 3 out of 4 business web sites are vulnerable to attack (Gartner)
- 75% of the hacks occur at the application level (Gartner)

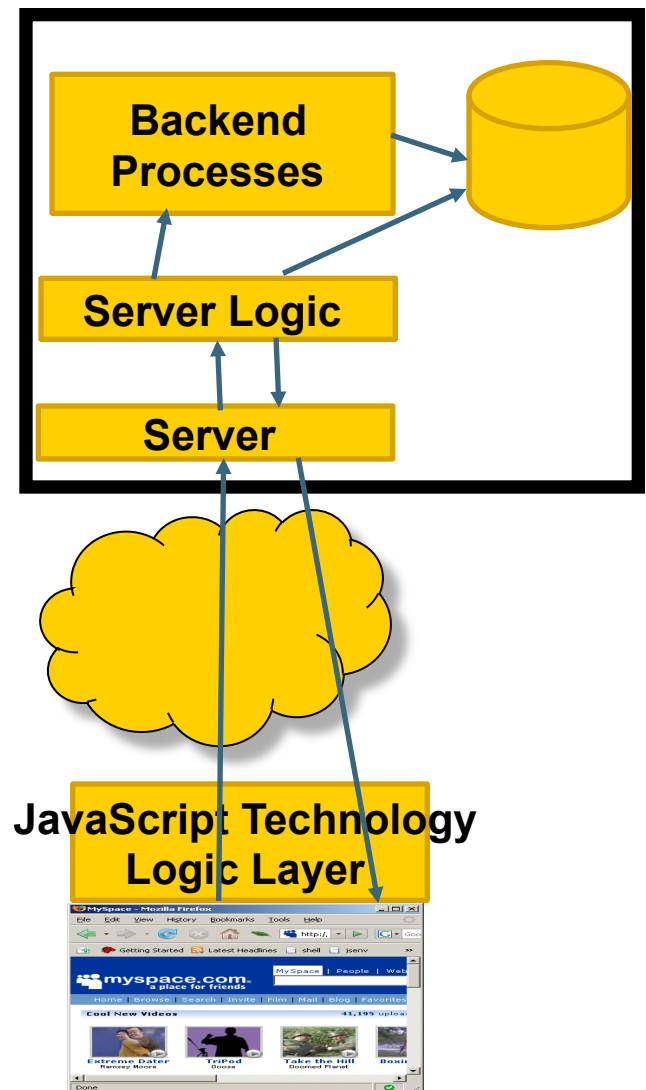


Source: Gartner



Architecture of Traditional Web Applications

- Browser – A thin client
- Most of the Application logic resides almost exclusively on server
 - Flow/business logic
 - Presentation logic
- Client acts as a dumb terminal sending actions to the server
- Server does all the processing and returns whole new page

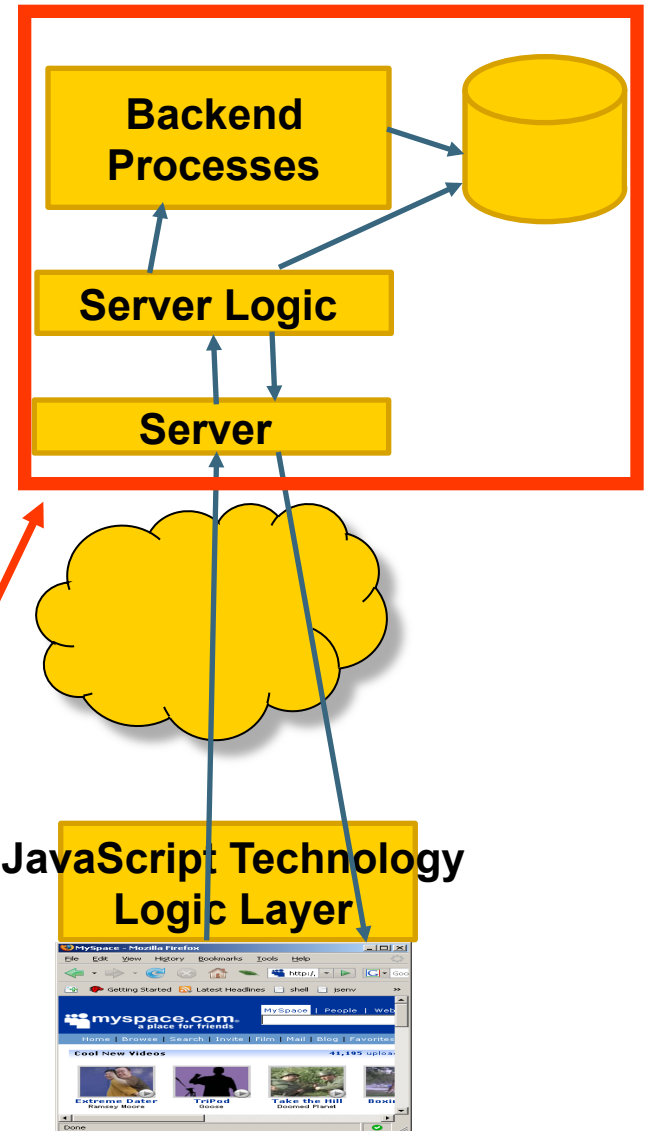


Attacks Against Traditional Web Applications

- Attacks involve
 - Sending malicious data
 - Sending code as data
 - Trying to access unauthorized data
- Malicious command hits edge cases in application design
- What did we say?
 - Validate input parameters
 - Encode output data
 - Use proper authentication
 - Use proper authorization



JavaScript Technology Logic Layer





JavaOne

Web Security – 2004 JavaOneSM Conference

JavaOne
Sun's 2004 Worldwide Java Developer Conference™

You Are Hacked ☹️
Ten Secrets to Securing Your
J2EE Web Applications

Karthik Shyamsunder
Selvamohan Neethiraj
Joel Nylund

VeriSign
The Value of Trust™

java.sun.com/javaone/sf

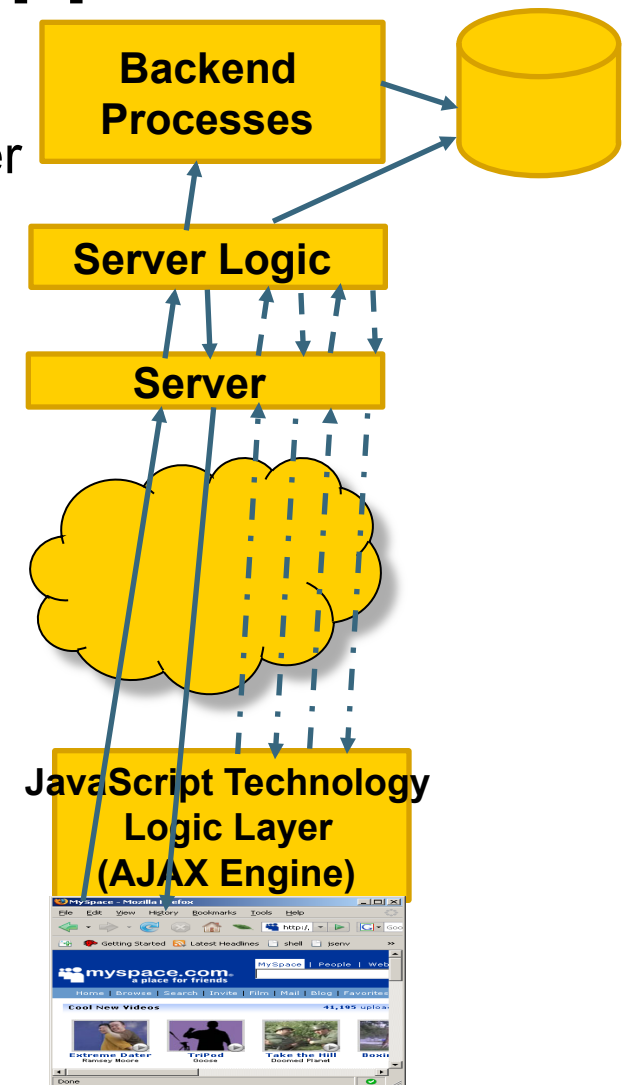
Java
Sun
microsystems

← | 2004 JavaOneSM Conference | Session TS-2137

- Download presentation from www.youarehacked.com

Architecture of an AJAX Application

- Browser - Rich/thick-client application
- Application logic resides both on client and server
- JavaScript technology takes on a bigger role
- Uses XMLHttpRequest object
- Fetch any kind of resource
 - HTML, GIF (view centric)
 - XML, JSON (data centric)
 - JavaScript technology (code centric)
- Client DOM tree is being manipulated

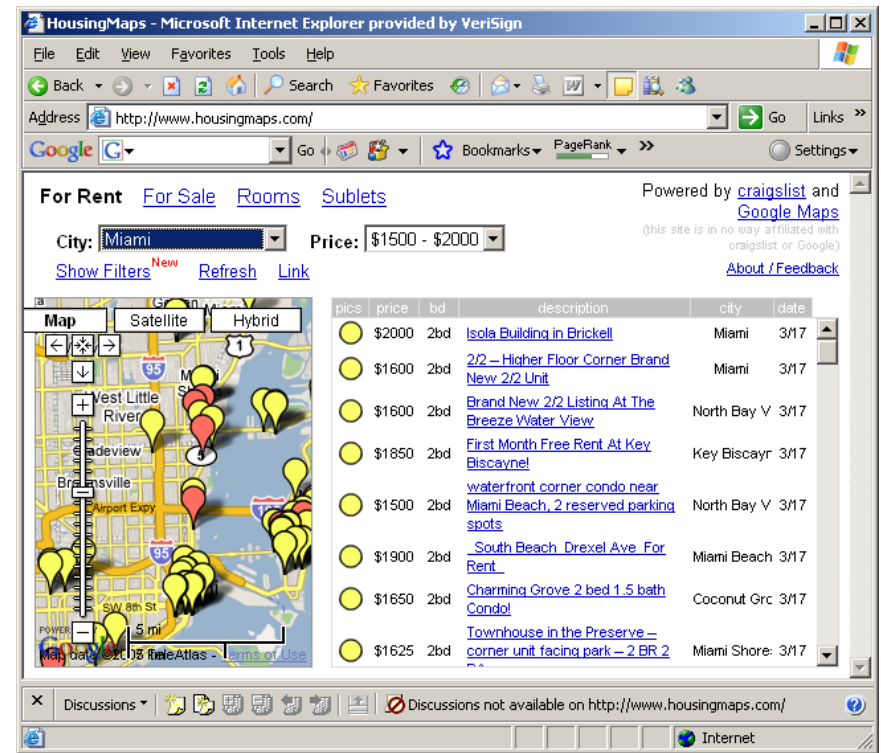
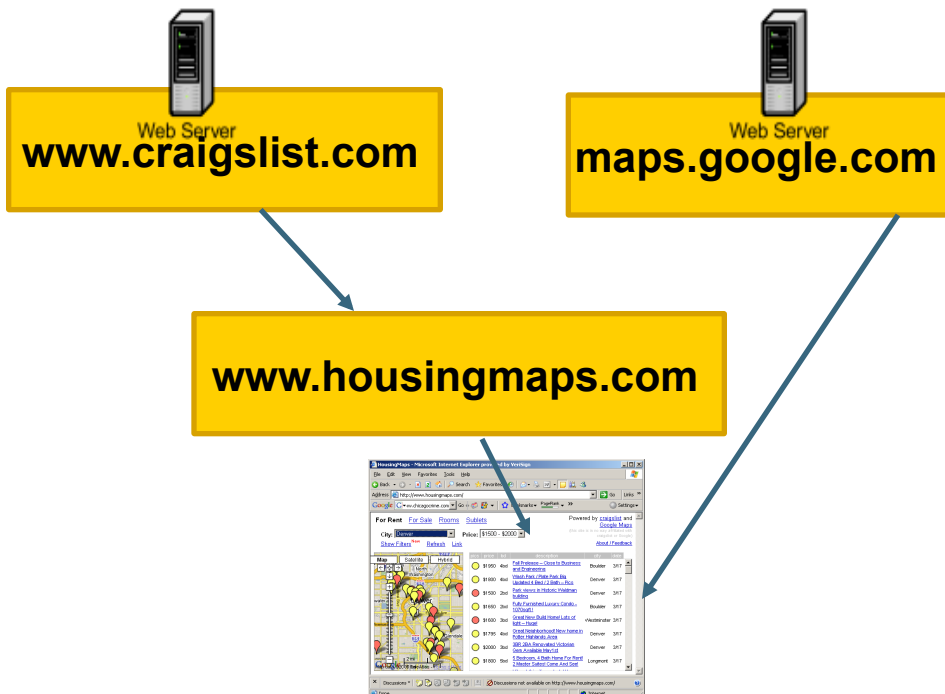




JavaOne

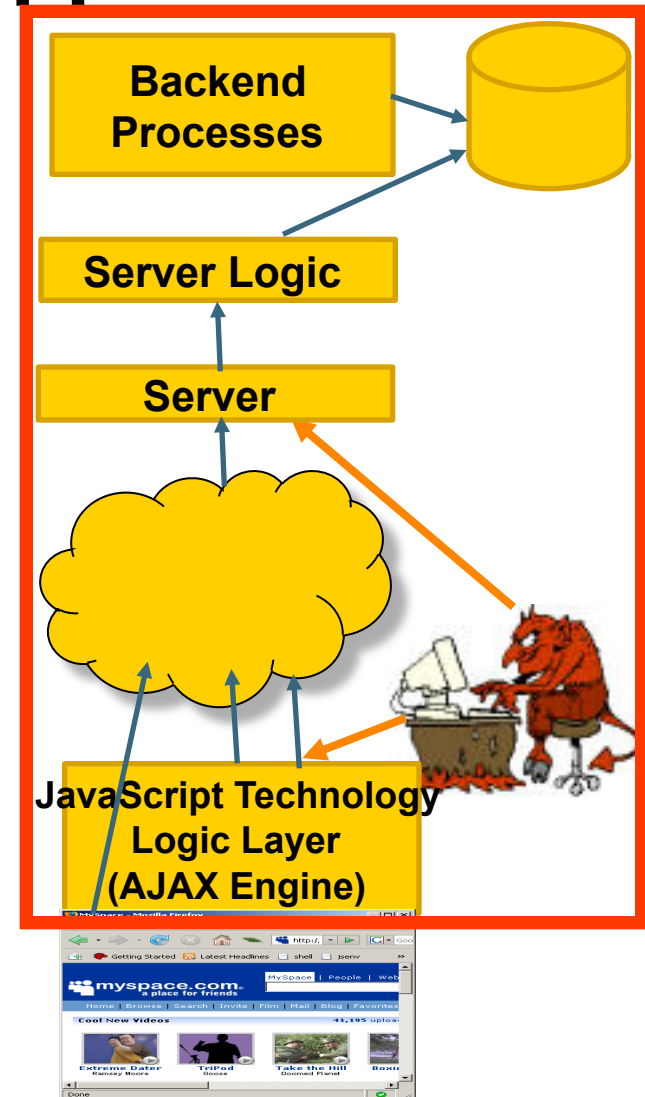
Web 2.0 Mashup Applications

- Aggregates services offered by other 3rd party applications to form a new application
- www.housingmaps.com
 - Mashup of craigslist & Google Maps



Attacks Against AJAX Applications

- Traditional web application attacks still apply
- Attacker is inside your application
 - Knowledge increases
 - Larger attack surface
 - Data serialization from unknown/untrusted sources
 - Companies migrate to AJAX without much thought to security
- In the case of mashups, attacking 3rd-party servers





Agenda

- Internet Threat Model
- **Browser Security Model**
- Vulnerabilities, Attacks, & Countermeasures
- Secure Software Development Process
- Summary
- Q&A



JavaOne

JavaScript Security in the Browser

- “Mobile code” = potential security risk
- Browsers execute JavaScript code in a sandbox
- Restrictions on JavaScript code in the sandbox
 - Cannot read/write files from/to the local system
 - Cannot execute any other programs
 - Cannot read the history of the browser
 - Cannot close a window that mobile code did not open
 - Cannot open a window that is too small





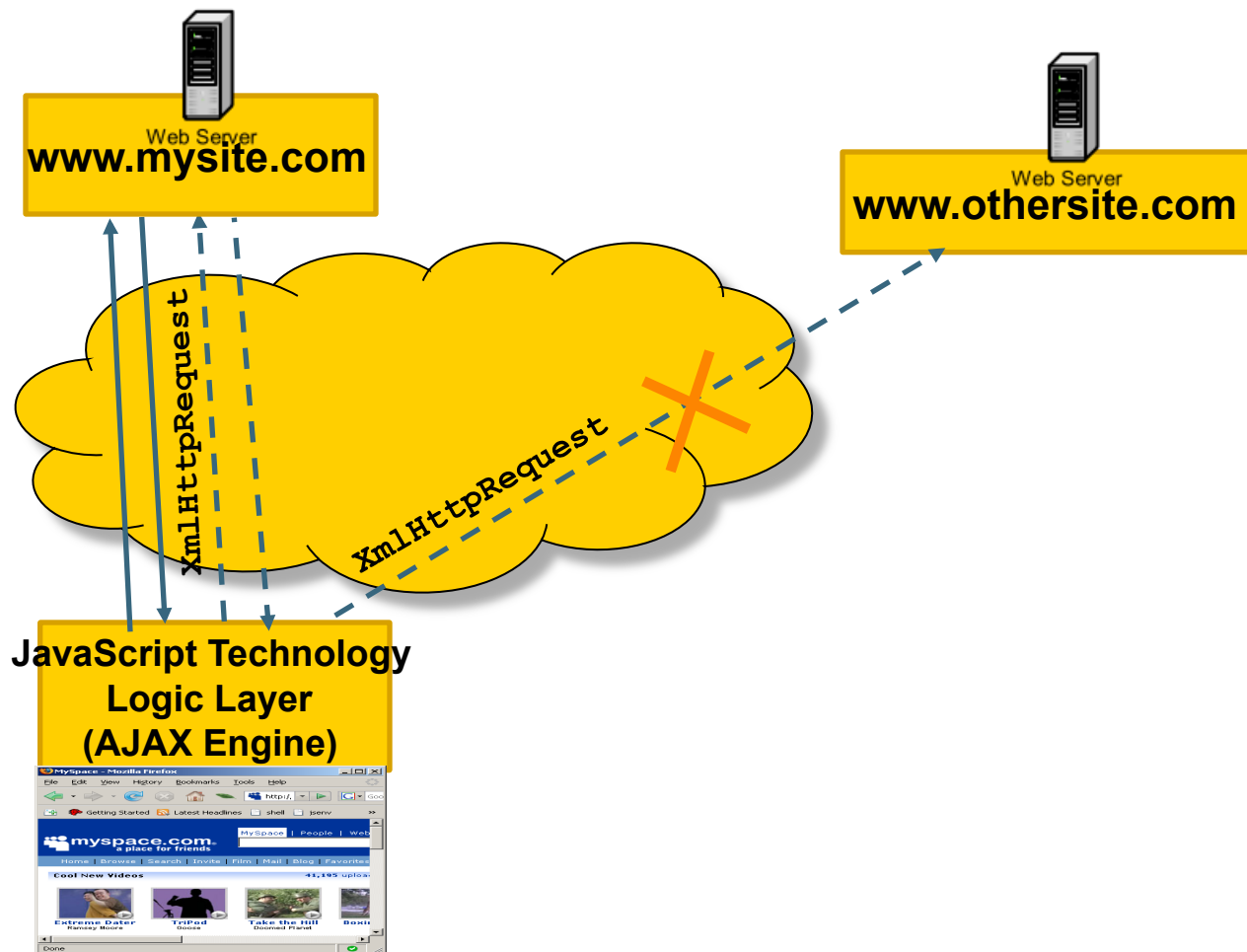
JavaOne

Browser's "Same Origin" Policy

- Also called "Server of Origin" Policy
- "Origin" = (protocol + host + port) parts of the URL
- Restriction was put to limit interaction between frames, iframes, and script tags from different origins
- Restriction extended to include XMLHttpRequest
 - Prevents client side JavaScript from making requests to any server other than the server from which it was downloaded
 - Different browser vendors implement this security somewhat differently



“Same Origin” Policy for AJAX





More “Same Origin” Policy Cases

	URLs	ALLOWED?	REASON
1	http://www.mysite.com/webapp1/action1	Yes	Although paths come from 2 different applications, the protocol, host and port is the same
	http://www.mysite.com/webapp2/action2		
2	http://www.mysite.com:8080/action1	No	Port numbers don't match
	http://www.mysite.com/action2		
3	http://www.mysite.com/action1	No	Protocols don't match
	https://www.mysite.com/actions2		
4	http://www.mysite.com/action1	No	Although www.mysite.com resolved to 128.220.101.100, but the browser does not work this out
	http://128.220.101.100/action2		
5	http://www.mysite.com/action1	No	Sub-domains are treated as separate domains
	http://scripts.mysite.com/action2		



JavaOne

Dancing Around the “Same Origin” Policy

1. Manipulate Browser Security Policy
2. Proxy Remote Services
3. Dynamic <SCRIPT> Tag

1. Manipulate Browser Security Policy

- Internet Explorer

- IE security system is based on the concept of “zones”
- Contacting external sites makes IE popup warning window
- IE trusts AJAX applications running from the local file system



- Mozilla

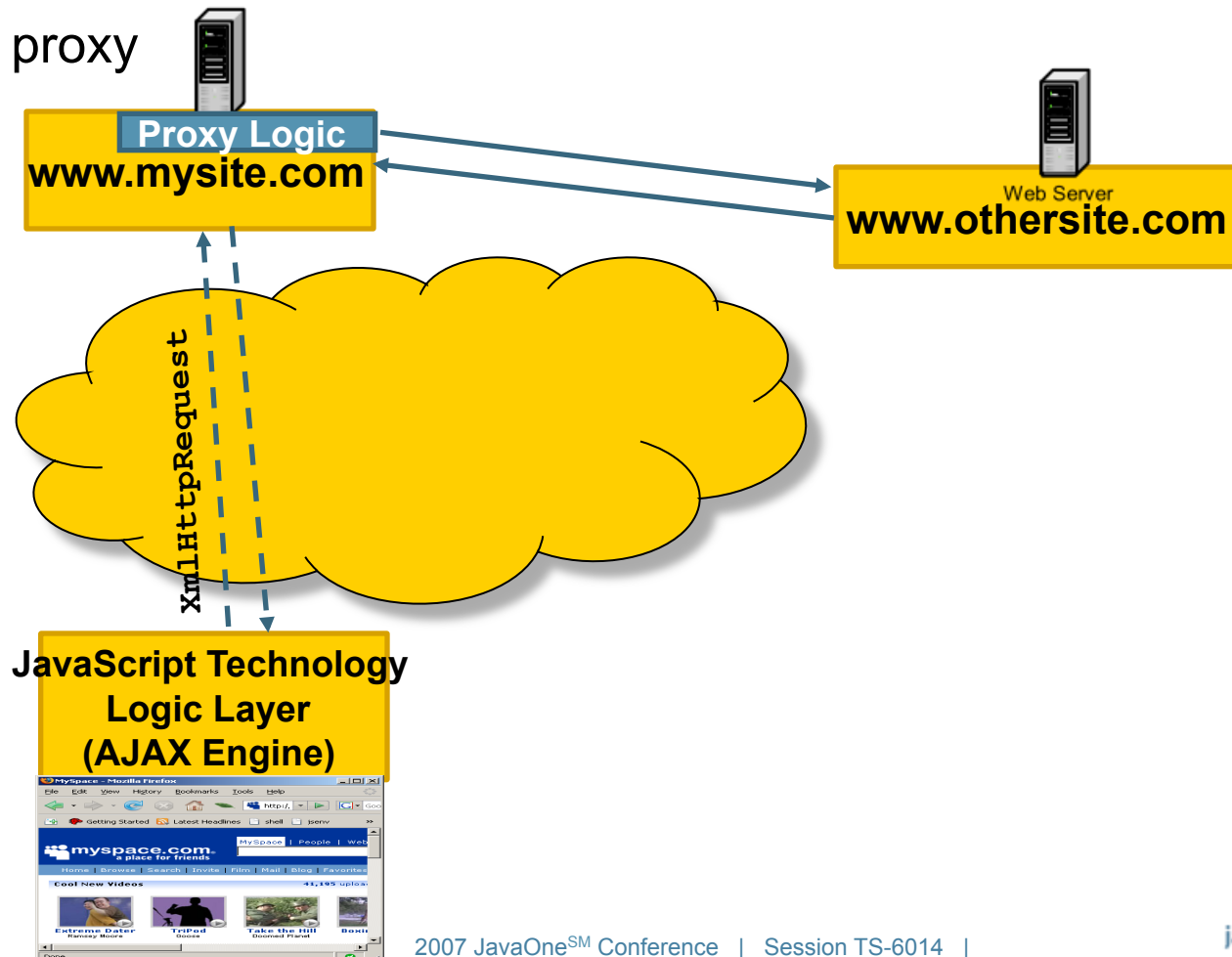
- Mozilla security system is based on the concept of privileges
- Application needs to request for privilege
- Privileges handled by netscape.security.PrivilegeManager
- To request privilege programmatically call enablePrivilege
- Firefox can be configured to not listen to privilege manager





2. Proxy Remote Services

- Also called “bridge” or “server-side proxy”
- 3rd-party proxy such as Apache mod proxy
- Custom proxy





3. Dynamic <SCRIPT> tag

- Create dynamic <SCRIPT> HTML tag instead of XMLHttpRequest
- Assign the **src** attribute the URL of the web service
- Append the <SCRIPT> to the page, which triggers the request
- Server returns JavaScript (or JSON object) executed in the browser

```
function yahooSearch() {  
    var head = document.getElementsByTagName("head").item(0);  
    var script = document.createElement("SCRIPT");  
    script.setAttribute("type", "text/javascript");  
    script.setAttribute("src", "http://api.search.yahoo...");  
    head.appendChild(script);  
}  
function yahooCallback(obj) {  
}
```

<http://api.search.yahoo.com/ImageSearchService/V1/imageSearch?appid=YahooDemo&query=madonna&output=json&callback=yahooCallback>



Agenda

- The Internet Threat Model
- Browser Security Model
- **Vulnerabilities, Attacks, Countermeasures**
- Secure Software Development Process
- Summary
- Q&A

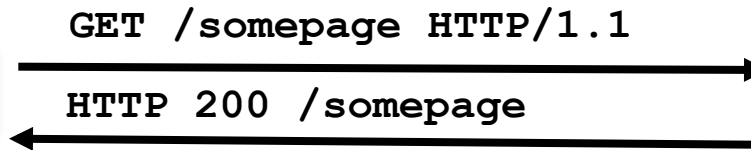


JavaOne

Exposure of Internal Details

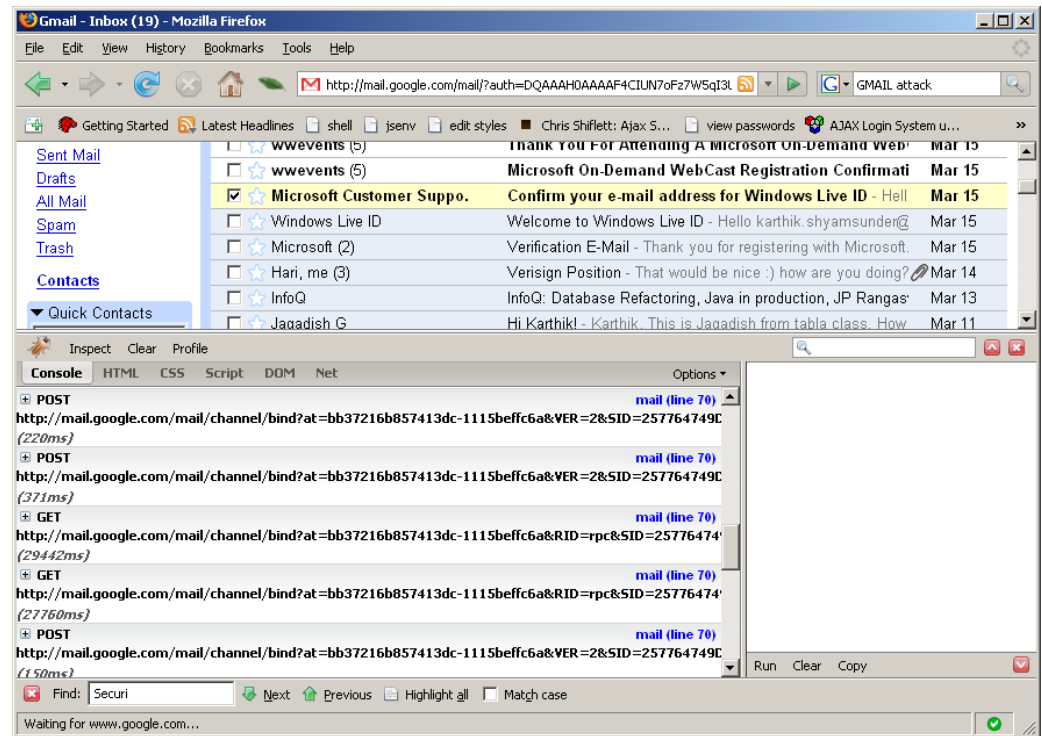
Vulnerabilities

- Profiling



Web Server

- What's new in Web 2.0?
 - Better Tools
 - Firebug
 - View DOM tree
 - Put breakpoints
 - Alter values
 - Watir
 - Ruby-based tool
 - Selenium
 - Java-based Tool





JavaOne

Exposure of Internal Details

Vulnerabilities

- What's new in Web 2.0?
 - Much more client-side code for hacker to view and dissect
 - Potentially more client-side comments for hacker to view
 - Better social community (blogs, newsgroups, forums)
 - Hackers' knowledge has increased
 - Application architecture/design details
 - Program business/logic flow details
 - Function names, variable names, return types
 - Helps build a footprint of the web application
 - Direct API access
 - Developers encouraged to expose more web services
 - Attacker calls your backend functions directly
 - Bypasses logic in the client side
 - Calls functions out of order





JavaOne

Exposure of Internal Details

Countermeasures

- Do not give out unnecessary information
- Remove comments from HTML/JavaScript technology code
 - Developer names, design details, notes, build numbers
 - Use build-time tools to remove comments
- Turn off WSDL for your web services
 - Many tools auto generate WSDLs – turn them off
 - No need to expose all services, inputs, & types to users





Exposure of Internal Details

Countermeasures

- Is AJAX the appropriate technology?
 - Use traditional web-application technology where security is a high priority
- Obfuscate your JavaScript technology code

```
function helloWorld(name) {  
    alert("Hello World " + name + "!");  
}
```

```
eval(function(p,a,c,k,e,d){while(c--){if(k[c]){p=p.replace(new  
    RegExp('\b'+c+'\b','g'),k[c])}}return p}('5 3(0){2("4 1  
    "+0+"!")}',6,6,'name|World|alert|helloWorld|Hello|  
    function'.split('|'))
```

- Note: obfuscation is not fool-proof



JavaOne

Cross-Site Scripting

Vulnerabilities

- Accomplished by code injection (HTML, JavaScript technology)

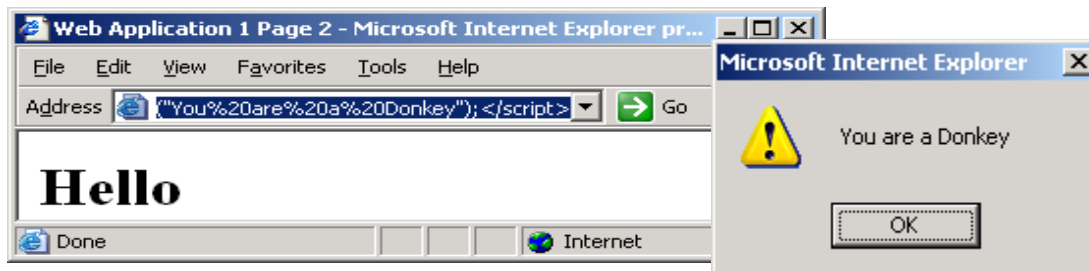
```
http://www.hackmebank.com/welcome.jsp?name=john
```

```
...  
<h1>Hello <%= request.getParameter("name") %></h1>  
...
```

`http://www.hackmebank.com/welcome.jsp?name=<i>CSS%20Vulnerable</i>`



`http://www.hackmebank.com/welcome.jsp?name=<script>alert("You%20are%20a%20Donkey");</script>`





JavaOne

Cross-Site Scripting Vulnerabilities

- Presentation/View Poisoning
 - Attacker does not attack the logic
 - Manipulates the CSS objects
 - Changes labels, re-skinning and repositioning UI components
- SCRIPT Injection
 - Injects malicious <SCRIPT> tag
 - New scripts
 - Invoke back-end functions
 - Make existing functions invalid

Please complete all of the information.

From:

To:

Amount: \$

Frequency:





JavaOne

Cross-Site Scripting

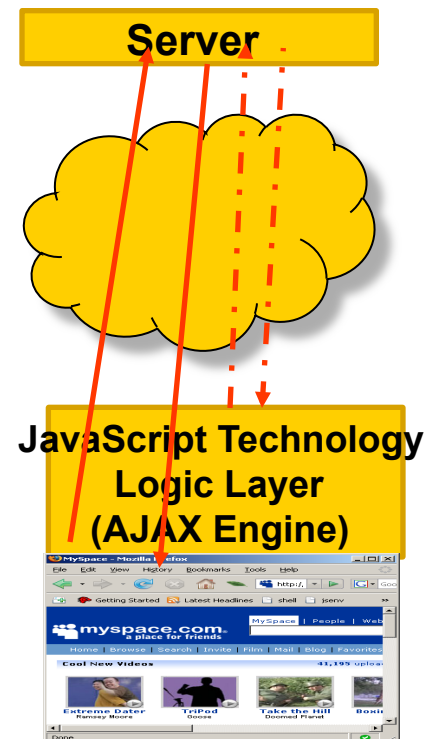
Countermeasures

- Practice input validation!
- Practice output encoding
 - HTML encoding when sending output to browser to avoid XSS
 - Practice JavaScript technology encoding to neutralize XSS



Cross-Site Request Forgery Vulnerabilities

- Also known as XSRF and CSRF and Cross-Site Reference Forgery
- Works by exploiting a trust that a user has in the application
 - ``
 - `<script src="http://host/command">`
 - `<iframe src="http://host/command">`
 - ```
<script>
 var foo = new Image();
 foo.src = "http://host/command";
</script>
```
- What's new Web 2.0?
  - Use `XMLHttpRequest` object to perform CSRF requests
  - Exposed web services amplify this attack
  - Better control over the request that can be sent
    - Can send HTTP headers, and make GET/POST request
    - Can receive HTTP status code, headers, and response data





# Cross-Site Request Forgery

## Countermeasures

- Common Misconceptions About Cross-Site Request Forgery
  - It is only exploitable in browser based applications
    - Scripts embedded into Word, Flash, Movie, RSS, or Atom web feed
  - It is not exploitable in POST based services
    - `<FORM>` tag can be used to submit POST requests
  - It can be prevented by implementing **Referer** header checking
    - **Referer** header can be spoofed by using `XMLHttpRequest`
  - It can be prevented by using the “one time token” pattern
    - Attacker can use existing XSS flaw to grab the token
- Potential Solutions
  - Implement POST-based service & **Referer** header checking & token approach
  - Prompt the user with PIN or strong CAPTCHA before each important action
  - Set a short time period for user sessions
  - Prevent XSS flaw



JavaOne

# Improper Validation

## Vulnerability

- Application accepts invalid/malicious input
  - SQL Injection, XSS, Parameter Tampering
- What's new in Web 2.0?
  - Validation confusion
    - Where is the validation done (client/server/both)?
    - With Sophisticated drag and drop IDE's, validation details are hidden
  - Complexity of data has increased
    - Lack of good toolkits/regular expressions available to validate these types of input
  - What input gets validated?
    - Developers usually validate GET/POST parameters
    - Developers often forget about HTTP Headers
    - Developers forget about file input (images, audio, video)
  - Trusting data from B2B partners
    - Mashups are bringing data from non-validated sources

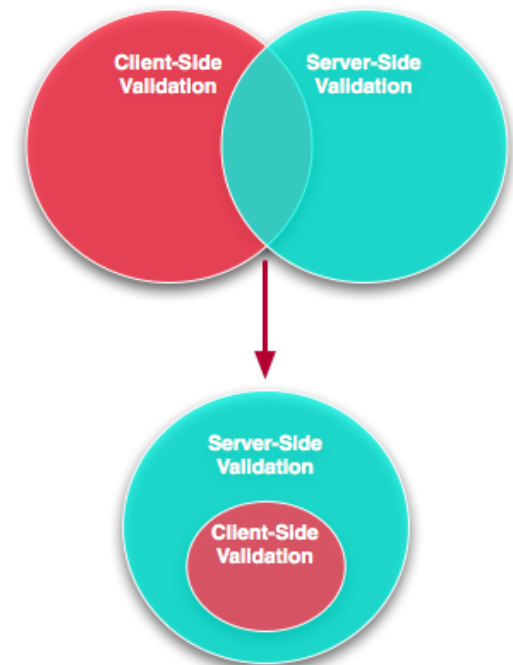




# Improper Validation

## Countermeasures

- Never trust the client!
- Validate all input data to the application
- Use strong validation techniques
  - Correctness, type, format, length, range, & context
  - Use white-listing instead of black-listing
  - Escaping input if possible
- Always validate on the server side
  - server-side validation = data integrity & security
- Client side validation as a subset of server side
  - client-side validation = usability & performance
- For mashups, never trust the external server





# Exploit Broken Authentication Vulnerabilities

Copyright 2002 by Randy Glasbergen. [www.glasbergen.com](http://www.glasbergen.com)

- Authentication
  - Act of proving who you say you are
  - Methods
    - User Name and Password
    - Certificate
- Broken Authentication leads to
  - Identity theft
  - Session hijacking
  - Loss of data
- Attack types
  - Man in the middle attack
  - Replay attack

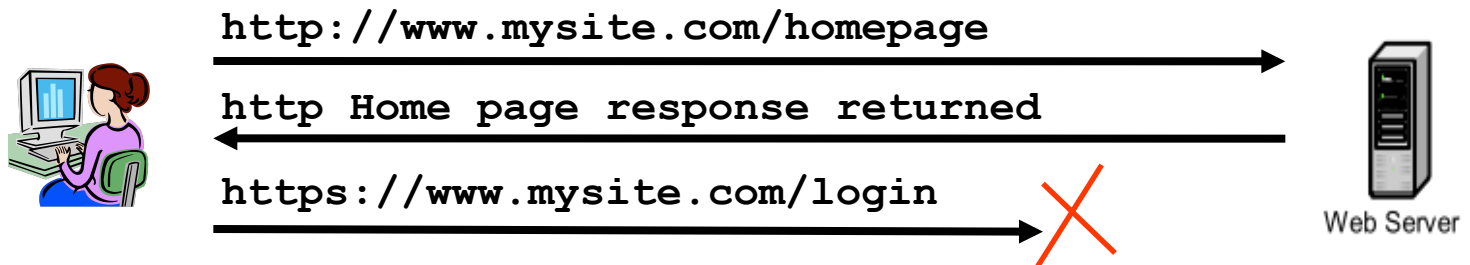


**“Someone got my Social Security number off the internet and stole my identity. Thank God — *I hated being me!*”**

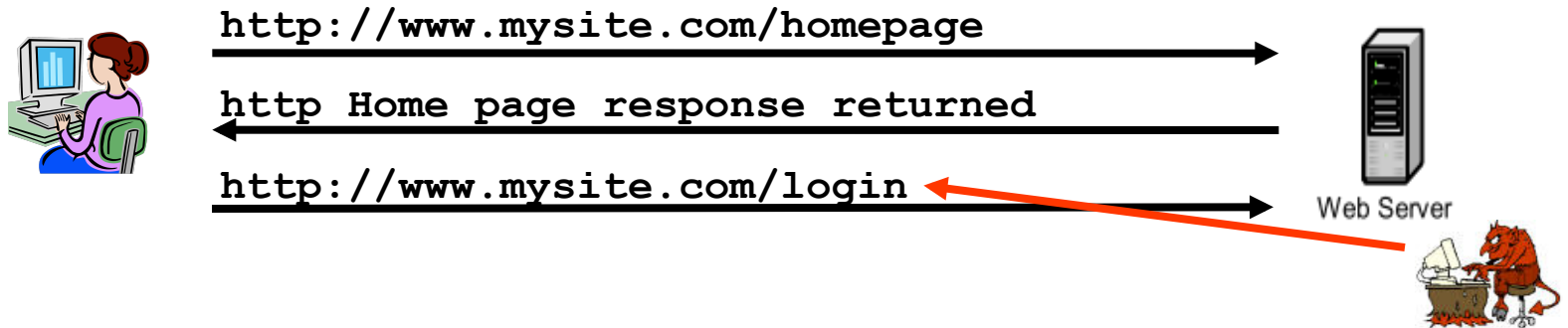


# Exploit Broken Authentication Vulnerabilities

- What's new in Web 2.0?
  - Most Web 2.0 applications are HTTP-based community sites
  - Scenario - HTTP AJAX application with HTTPS authentication



- What do developers typically do?
  - Use HTTP for entire AJAX application





JavaOne

# Exploit Broken Authentication

## Countermeasures

- Option 1: Use HTTPS for the entire Web 2.0 application
  - Does address the “Same Origin” Policy
  - Hacker can not sniff any packets
  - Frequent SSL handshakes is expensive
    - Full or partial SSL handshake is dependent on the browser



`https://www.mysite.com/homepage`

Home page response returned

`https://www.mysite.com/login`

Logged in page



Web Server



# Authentication Issues

## Countermeasures

- Option 2: Use HTTP with the “Direct Login” AJAX pattern
  - Addresses the “Same Origin” Policy
  - Does not incur the HTTPS cost
  - Requires encrypting password so it cannot be decoded or replayed
    - Use server one-time random challenge token
    - Use of double-hashed password



`http://www.mysite.com/homepage`

Home page with challenge token

Hash ((hash(password)) + challenge token)

`http://www.mysite.com/login`

Calculates double-hashed password and compare  
and Return Success



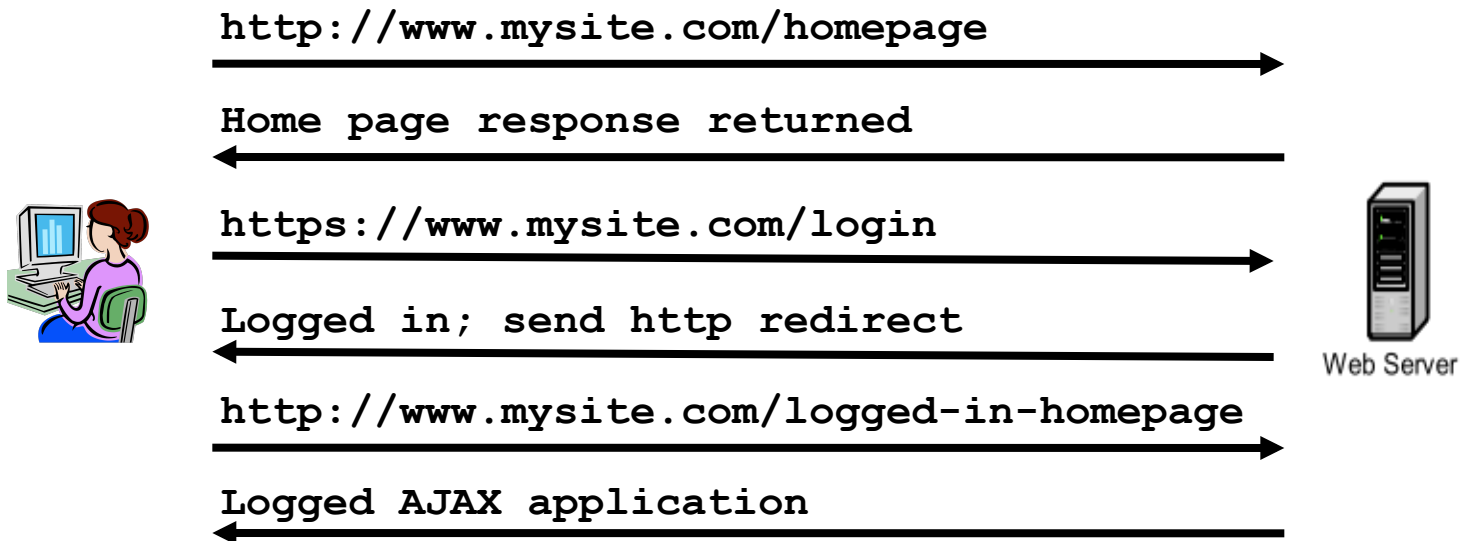
Web Server



# Authentication Issues

## Countermeasures

- Option 3: Use traditional HTTPS login page with redirect to HTTP AJAX application
  - Can use plain HTTP for AJAX application
  - Can use a secure transport when passing user credentials
  - Simpler than the “Direct Login” AJAX pattern





# Exploit Broken Access Control Vulnerabilities

- Access Control is also called Authorization
- Implementing correct access control is not trivial
- Broken Access Control leads to
  - Un-authorized access to sensitive data
  - Un-authorized users executing illegal transactions
- What's new in Web 2.0?
  - Authorization logic can be exposed in the client
    - Exposure of Role names
    - Exposure of unauthorized functions
    - Hacker can by-pass client-side authorization logic
  - Increased number of services that need to be protected
    - Hackers can obtain the services from the client-side code
    - Hackers can guess the names of services from the client-side code
    - Hackers can obtain the services from published WSDL's
    - Hackers can call services out-of-order



# Exploit Broken Access Control

## Countermeasures

- Minimize exposure of authorization logic in the AJAX client!
- Always check authorization on the server
- Java Platform security model still applies (declarative & programmatic)

```
<security-constraint>
 <web-resource-collection>
 ...
 <url-pattern>services/*</url-pattern>
 </web-resource-collection>
 <auth-constraint>
 <role-name>manager</role-name>
 </auth-constraint>
</security-constraint>
```

```
if (!request.isUserInRole("manager")) {
 throw new UnauthorizedException();
}
```



JavaOne

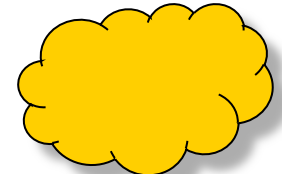
# Denial of Service

## Vulnerabilities

- Attack by which a hacker prevents legitimate users of a service from using that service
- Misconception is DOS and DDOS are primarily network-level attacks and not application attacks
- What's new in Web 2.0?
  - Attacks against the client:
    - Can tie up client host CPU with infinite loop
    - Can use `window.setInterval()` command to avoid being shutdown
    - Can take advantage of browser bugs
    - Can inject bad data to make browser crash



Mysite server



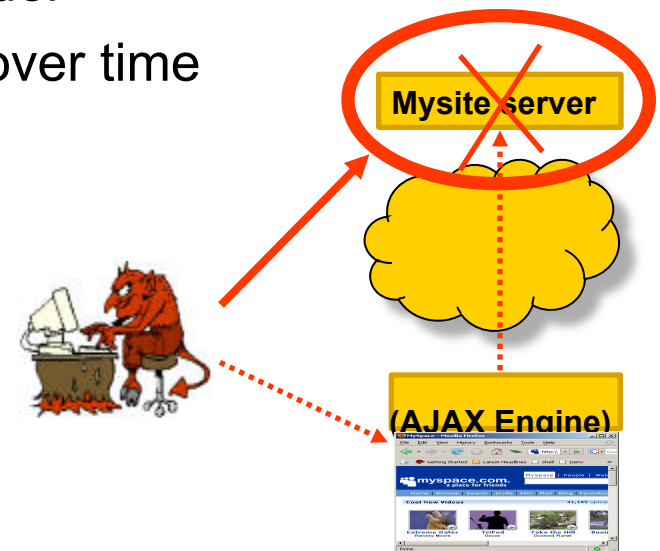


JavaOne

# Denial of Service

## Vulnerabilities

- What's new in Web 2.0? (continued)
  - Attacks against the server
    - Directly call exposed web services uncontrollably
    - Insert malicious scripts to invoke services uncontrollably
    - Call exposed web services out of order
    - Insert malicious scripts that spread over time
    - For example, Samy worm

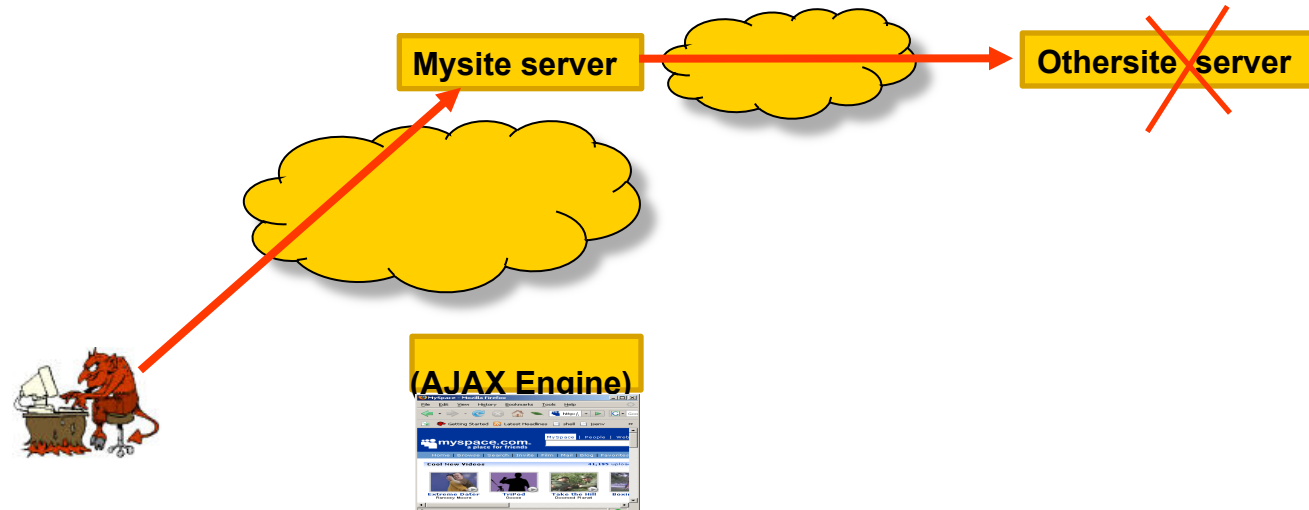




# Denial of Service

## Vulnerabilities

- What's new in Web 2.0? (continued)
  - Attacks against the server's server
    - Also called "smashing the mashup"
    - Attacks the proxy bridge to attack the "other site"
    - Takes advantage of trust between B2B
    - Takes advantage of the resource capabilities in B2B applications
    - Difficult for the "other site" to distinguish normal usage vs. an attack





# Denial Of Service

## Countermeasures

- Since most of the DOS is caused by code injection, cross-site scripting bugs need to be eradicated
  - Validate input data effectively
  - Practice output encoding
- Rate-limit requests in B2C and B2B services
  - Limit Number of requests in a given period (by minute, hour, day)
  - Implement Dynamic bandwidth limitation
    - Decrease bandwidth with increased volume
- Prevent automation
  - Use CAPTCHA schemes
  - Use “one time token” design pattern
- Have good monitoring, real-time analysis & alert systems in place
- Tune system for “at most” performance (CPU, JVM, network, ...)
- Over-provision your system



JavaOne

# Code Complexity Issues

## Vulnerabilities

- Already developers have to master a lot!
  - If the technology is not well understood the risk of introducing vulnerabilities increases
- What's new in Web 2.0?
  - Now there are even more technologies to master!
  - JavaScript technology is playing a bigger role
    - Dynamic language (RTTI, dynamic types, closures)
    - Dynamic code generation and execution using `eval()`
    - Most errors happen at runtime
    - Not only master new concepts, but how they interact
    - Thread-safety issues





JavaOne

# Code Complexity Issues

## Countermeasures

- Education Education Education!
  - Learn JavaScript technology!
  - Pass this security presentation around in your group
- Practice unit testing for JavaScript technology code too
- Understand issues related to AJAX concurrency
  - Both client and server side
  - Understand that browsers launch requests simultaneously
  - Potentially synchronize on session object in servlets
- Build time in schedule for developers to ramp up



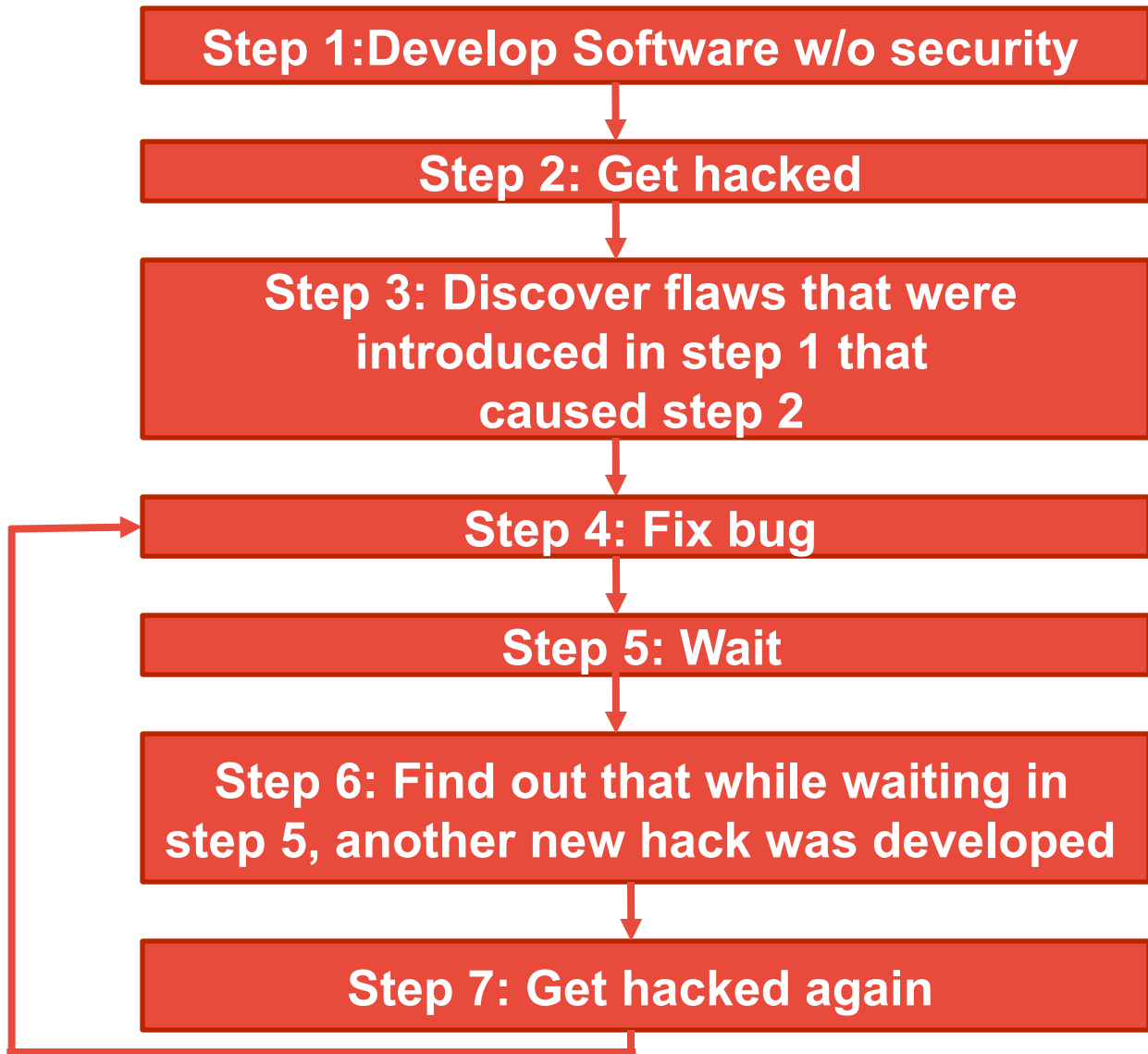


# Agenda

- Internet Threat Model
- Browser Security Model
- Vulnerabilities, Attacks, and Countermeasures
- **Secure Software Development Process**
- Summary
- Q&A



# Seven Steps of DOOM

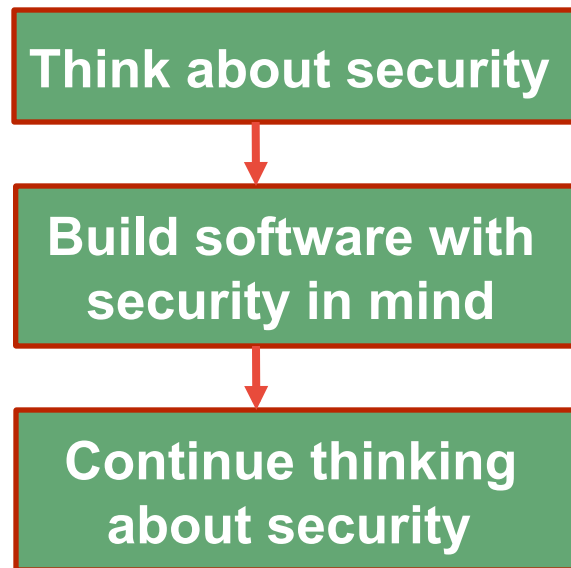




# Steps to Success

## Security in Software-Development Life Cycle

- Secure design
- Secure development
- Secure testing
- Secure deployment & operations
- Audit process





JavaOne

# Application Security Review

## Security in Software-Development Life Cycle

1. Identify assets
2. Create a security architecture
3. Identify and document vulnerabilities
4. Assess your risk
5. Plan for risk mitigation



# Summary

- AJAX is a powerful suite of technologies
  - AJAX can improve user experience
- But, be aware of security risks
  - Creates new ways to attack via old vulnerabilities
- Always **keep security in mind** when building applications



JavaOne

# Research Contributors

- **David Smith**
- **Engineer – VeriSign, Inc.**
- **dsmith@verisign.com**
  
- **Rajesh Badam**
- **Engineer – VeriSign, Inc.**
- **rbadam@verisign.com**
  
- **Satish Dandu**
- **Engineer – VeriSign, Inc.**
- **sdandu@verisign.com**





JavaOne

## For More Information

- <http://www.youarehacked.com>
- <http://www.owasp.org>
- <http://www.cgisecurity.com/>
- <http://www.webhackingexposed.com/>
  
- TS-6536
  - Enabling Identity 2.0 in Java Technology





# Q&A

Karthik Shyamsunder  
Principal Engineer  
VeriSign, Inc.

James Gould  
Principal Engineer  
VeriSign, Inc.