Security for Web Languages

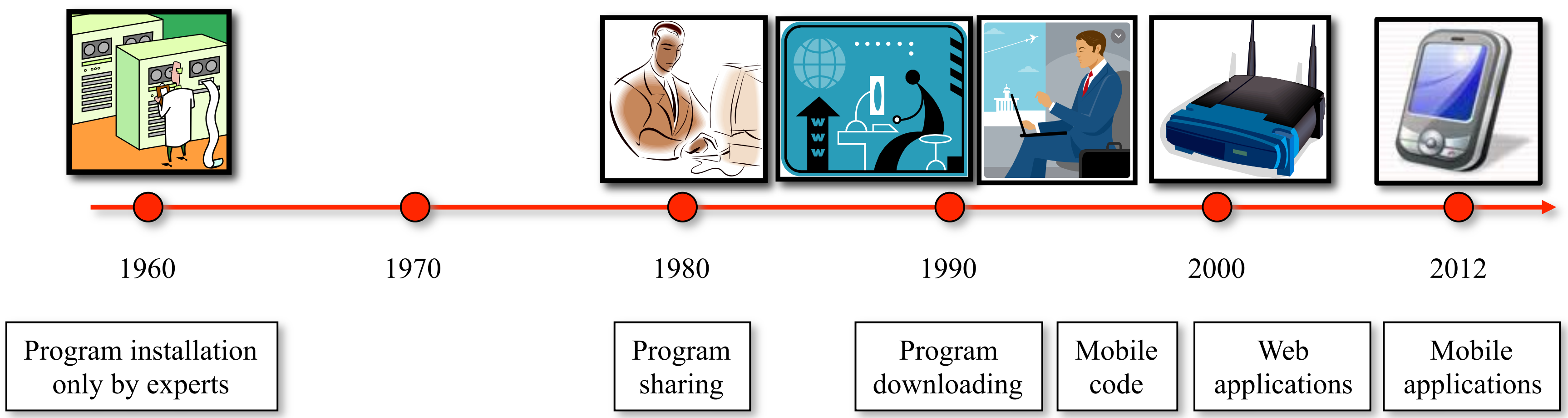
Marco Pistoia, Ph.D.
Manager, Research Staff Member
IBM T. J. Watson Research Center

New York University
July 19, 2012

Computer Security

- Hardware, software, and network security to prevent:
  - Service stealing
  - Denial of service
  - Confidentiality violations
  - Integrity problems
  - Service misuse
- Most of today's mechanisms are insufficient to guarantee computer security

Threat Evolution



Top-Ten Web-Application Security Vulnerabilities

Top-ten security vulnerabilities according to the Open Web Application Security Project (OWASP) (`http://www.owasp.org`)

1. Injection
2. Cross-site scripting
3. Broken authentication and session management
4. Insecure direct object reference
5. Cross site request forgery (CSRF)
6. Security misconfiguration
7. Unsecure cryptographic storage
8. Failure to restrict URL accesses
9. Insufficient transport-layer protection
10. Unvalidated redirects and forwards

The Need for Language-Based Security

- *Operating-system security* is low-level
- Many attacks are at the application level
- Operating-system security is insufficient
- *Language-Based Security* is the ability to define security policies and enforcement mechanisms using program analysis or techniques that are embedded into the programming language
- Enforcement time:
  - Before: Analyze and fix
  - During: Monitor and halt
  - After: Roll back

Outline

- Fundamental security concepts and principles
  - Access control
  - Information security
  - Principle of Least Privilege
  - Principle of Complete Mediation
- Analysis for access control and information flow

Part I

Fundamentals Security Concepts and Principles

Access Control

- Mechanism to define and enforce which principals can access which resources
- Two components:
  - *Authentication* ascertains the identity of the principal who is making the requests
  - *Authorization* verifies that the principal is allowed to access the resource that has been requested

Authorization Decisions and Authorization Matrix

- An authorization decision can be seen as a function
- An authorization policy can be seen as a matrix [Lampson, 1992]
  - The columns of the matrix are Access Control Lists (ACLs)
  - The matrix grants access to system resources to users and groups

(*principal*, *request*, *object*) → *true*/*false*

	File `C:\log.txt`	**Socket `ibm.com:80`**	**System configuration**
Administrator principal	`read, write, execute`	`listen, connect`	`read, write`
Text editor program	`read, write`	`-`	`read`
Internet browser	`-`	`connect`	`read`

Role-Based Access Control (RBAC)

- RBAC is a form of access control that can better represent the protection of information in enterprise systems [Ferraiolo and Kuhn, 1992]
- A role is a set of permissions
- Each permission represents a responsibility in an enterprise
- Roles are then assigned to users and groups

The Principle of Least Privilege

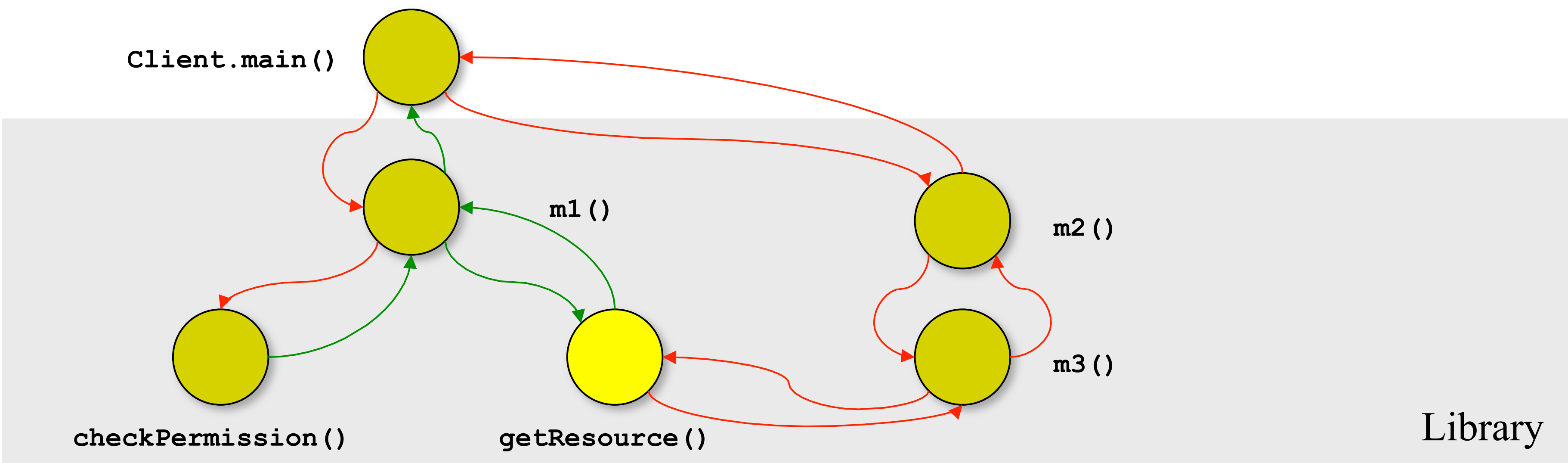
- In a computing environment, every module (such as a process, a user, or a program) must be able to access only such information and resources that are necessary to its legitimate purpose [Saltzer and Shroeder, 1975]
- Example:
  - Grant a text editor the permission to access the file system
  - Do not grant a text editor the permission to open a socket connection

Problems in Enforcing the Principle of Least Privilege

- An authorization policy must be neither too permissive nor too restrictive
  - Too permissive:
    - Violation of the Principle of Least Privilege
    - Program exposed to security attacks
  - Too restrictive
    - The policy-enforcement mechanism will generate run-time authorization failures
    - Security problems may arise

The Principle of Complete Mediation

- Every access to any resource must be mediated by an appropriate authorization check [Saltzer and Shroeder, 1975]



Problems in Enforcing the Principle of Complete Mediation

- Enforcement is system-specific
  - Different systems have different resources that need to be protected
  - Different systems have different protection mechanisms
- The authorization check for a particular resource must check for authorization appropriately
- Authorization caching can cause violations of the Principle of Complete Mediation

Information Security

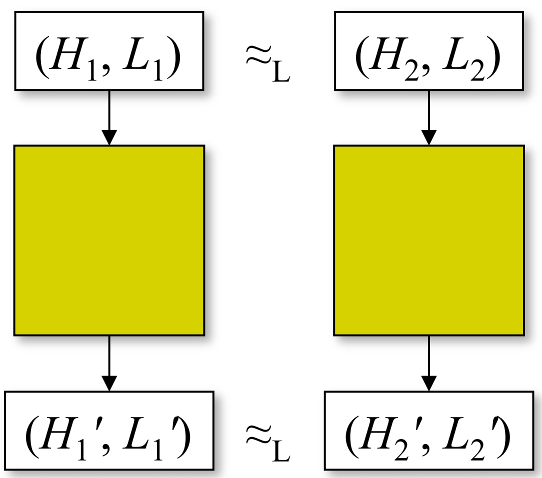
- No illicit flow of information should be allowed in a program
- Two dimensions of information security:
  - *Integrity*: Valuable information should not be damaged by any computation
  - *Confidentiality*: Valuable information should not be revealed by any computation
- Confidentiality different from:
  - *Secrecy*: Secret information is not leaked to public listeners
  - *Anonymity*: A public observer cannot learn the identities of the participating principals even though actions might be known

Static Information Flow

- Information-flow policies are partial orders [Denning, 1976]
- Programs are annotated with integrity and confidentiality information-flow policies [Denning and Denning, 1977]
- The compiler
  - Verifies that all the execution of the program satisfy the policies
  - Transforms the program to ensure that policies are obeyed
- The run-time system validates the program policies against the system policies

Noninterference

- “Low behavior of the program is not affected by any high security data” [Goguen and Meseguer, 1982]
- Dual interpretation for integrity and confidentiality



$$L_1 = L_2$$

$$L_1' = L_2'$$

	Integrity	**Confidentiality**
High	Untrusted	Secret
Low	Trusted	Public

Security Types

- Add information-flow policies as type annotations
- Reject any flow from higher to lower
- Proving noninterference
  - Any type-safe program with information-flow security types satisfies noninterference [Volpano, et al, 1996]
  - Proved by showing that each execution step preserves low-observable equivalence

Java Information Flow (Jif)

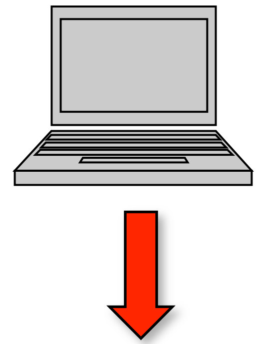
- Jif [Myers, 1999] annotates Java programs with labels
  - A label contains a policy in terms of principals
  - A variable has a type and a label
- Achieves both access control and information flow

Downgrading

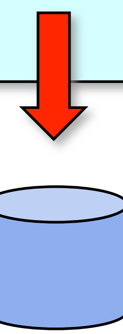
- An information-security policy can establish that:
  - Certain parts of secret information can be *declassified* and revealed to certain public listeners. For example:
    - Last 4 digits of SSN can be revealed to bank teller
    - Result of a password check can be revealed to anyone
  - Certain parts of untrusted input can be *endorsed* and used in certain trusted computations. For example:
    - Untrusted user input can be used in a Web application if it is properly formatted

	Integrity	**Confidentiality**
High	Untrusted	Secret
Low	Trusted	Public
Downgrading	Endorsement	Declassification

Example: Injection Flaws in Web Applications



```
public void submitQuery(String userName) {
    String query =
       "SELECT id FROM users WHERE name = '" + userName + "'";
    execute(query);
}
```



Problems in Enforcing Information Security

- Policies can become very complex
- It may be difficult and expensive to track the actual flows of information
  - Complex flows through the program
  - Covert channels
- Implicit flows
  - Confidentiality: value of `x` may reveal values of `a` and `b`
  - Integrity: value of `b` influences value of `x` even if `b` is `false`

```
int x = 0;
if (b) {
   x = a;
}
```

Part II

JavaScript Security

Risks with JavaScript

- Downloading and running programs written by unknown parties is dangerous
- Most people do not realize that nearly every time they load a Web page, they are allowing code written by an unknown party to execute on their computers
- Since it would be annoying to have to confirm your wish to run JavaScript each time you load a new Web page, browsers implement a security policy designed to reduce the risk such code poses to the end user
- Example: JavaScript code cannot access your file system

JavaScript Security Model

- Scripts are confined inside a sandbox where they cannot have access to the operating system or file system
- Scripts are permitted access only to data in the current document or *closely related documents* (those from the same site as the current document)
- No access is granted to the local file system, the memory space of other running programs, or the operating system's networking layer

The Reality

- The reality of the situation, however, is that often scripts are not properly sandboxed
- There are numerous ways that a script can exercise power beyond what you might expect, both by design and by accident
- The fundamental premise of browsers' security models is that there randomly encountered code is by default hostile
- However
  - Code coming from trusted sources can escape the sandbox, often without requiring the explicit consent of the user
  - Scripts can gain access to otherwise privileged information in other browser windows when the pages come from related domains

Same-Origin Policy

- It is the primary JavaScript security policy
- It prevents scripts loaded from one Web site from getting or setting properties of a document loaded from a different site or using a different protocol and port number
- It applies to scripts attempting to access the content of frames
  - If two frames have not been loaded from the same site using the same protocol, scripts cannot cross the framed boundary

Same-Origin Check

- When a script attempts to access properties or methods in a different window, for example, using the handle returned by window.open(), the browser performs a same-origin check on the URLs of the documents in question
  - If the URLs of the documents pass this check, the property can be accessed
  - If they do not, an error is thrown
- The same-origin check consists of verifying that the URL of the document in the target window "has the same origin" as the document containing the calling script
- Two documents *have the same origin* if they were loaded from the same server using the same protocol and port

Problems

- Older browsers did not enforce the same-origin policy correctly
- The same-origin policy does not protect against cross-site interactions when two Web sites are hosted by the same server
- You cannot turn off the same-origin policy, for example in an intranet, so you have to use ActiveX controls or use signed scripts
- Denial of service attacks are possible

Example

A JavaScript program was loaded from `http://www.nyu.edu/dir/page.html`

	URL of Target Window	Result	Motivation
1	`http://www.nyu.edu/index.html`	success	
2	`http://www.nyu.edu/~hirzel/index.html`	success	
3	`ftp://www.nyu.edu`	failure	Different protocol
4	`http://www.columbia.edu/index.html`	failure	Different domain
5	`http://www.nyu.edu:80/index.html`	success	
6	`http://www.nyu.edu:8080/index.html`	failure	Different port
7	`http://www2.nyu.edu/dir/page.html`	failure	Different domain

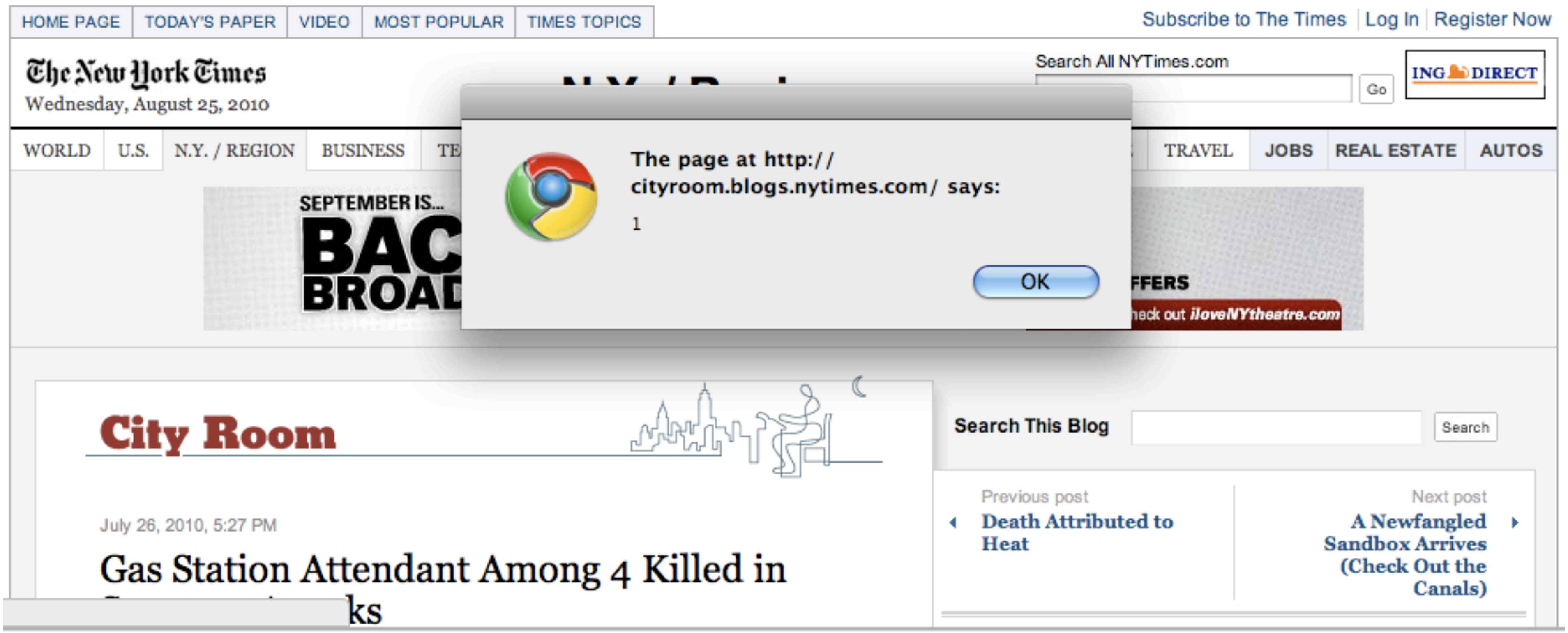
XSS

- Consider a site that accepts a user name in form input and then displays it in the page
- Entering the name John and clicking **Submit** might result in loading a URL like http://www.example.com/mycgi?username=John, and the following snippet of HTML to appear in the resulting page:
  Hello, John!
- If someone can get you to click on a link to http://www.example.com/ mycgi?username=John<script>alert('Uh oh');</script>, the CGI might write the following HTML into the resulting page:
  Hello, John<script>alert('Uh oh');</script>
- The script passed in through the username URL parameter was written directly into the page, and its JavaScript is executed as normal

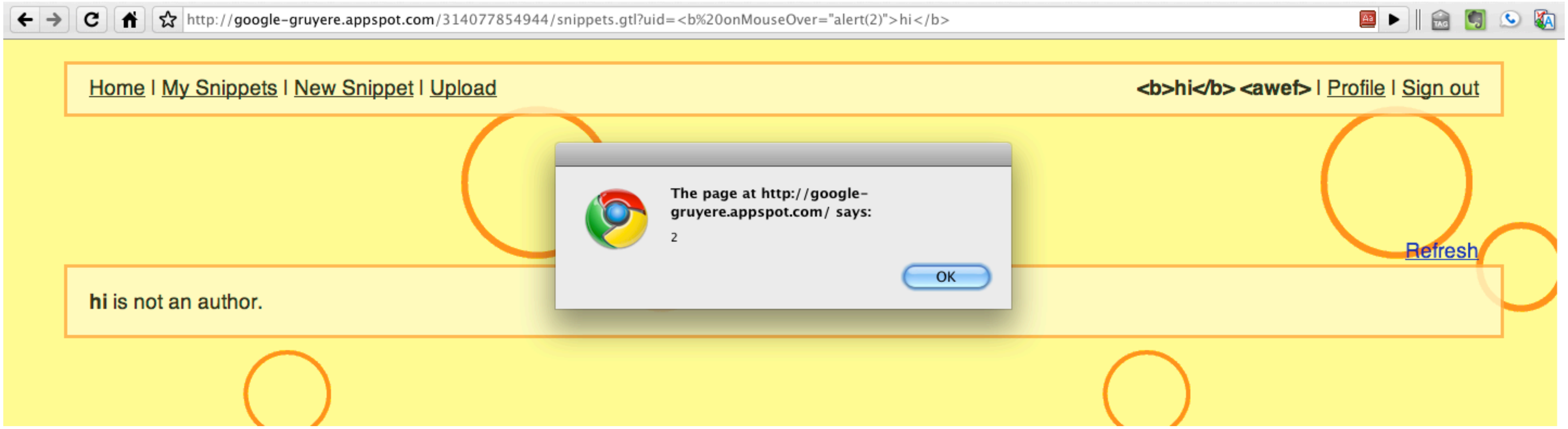
XSS Prevention

- Input validation
- HTML-escape data


ajax.googleapis.com/ajax/libs/swfobject/2.1/swfobject.js
/* SWFObject v2.1 <http://code.google.com/p/swfobject/>
Copyright (c) 2007-2008 Geoff Stearns, Michael Williams, and Bobby van der Sluis
This software is released under the MIT License <http://www.opensource.org/licenses/mit-license.php>
*/



HOME PAGE
TODAY'S PAPER
VIDEO
MOST POPULAR
TIMES TOPICS
Subscribe to The Times
Log In
Register Now
The New York Times
Wednesday, August 25, 2010
Search All NYTimes.com
Go
ING DIRECT
WORLD
U.S.
N.Y. / REGION
BUSINESS
TRAVEL
JOBS
REAL ESTATE
AUTOS
SEPTEMBER IS...
The page at http://
cityroom.blogs.nytimes.com/ says:
1
OK
City Room
Search This Blog
Search
Previous post
Death Attributed to Heat
Next post
A Newfangled Sandbox Arrives (Check Out the Canals)
July 26, 2010, 5:27 PM
Gas Station Attendant Among 4 Killed in



Consequences of Taint Violations

- Read and write access to saved data in cookies and local data stores
- Read and write access to data in the web page
- Key loggers
- Impersonation
- Phishing via page modifications or redirects

Getting data from the DOM

Sanitizing some, but not all, of the data

```
var el1 = document.get    entById("d1")
function foo() {
  var el2 = document.getElementById("
  function bar() {
    var el3 = new Element();
    var s = encodeURIComponent(el2.innerText);
    document.write(s);
    el1.innerHTML = el2.innerText;
    document.location = el3.innerText;
  }
  bar();
}
foo();
function baz(a, b) {
 a.f = document.URL;
 document.write(b.f);
}
var x = new Object();
baz(x, x);
```

Writing untrusted data into web page

Writing unchecked data to the web page

```
var el1 = document.getElementById("d1");
function foo() {
  var el2 = document.getElementById("d2");
  function bar() {
    var el3 = new Element();
    var s = encodeURIComponent(el2.innerText);
    document.write(s);
    el1.innerHTML = el2.innerText;
    document.location = el3.innerText;
  }
  bar();
}
foo();
function baz(a, b) {
 a.f = document.URL;
 document.write(b.f);
}
var x = new Object();
baz(x, x);
```

```
var el1 = document.getElementById("d1");
function foo() {
  var el2 = document.getElementById("d2");
  function bar() {
    var el3 = new Element();
    var s = encodeURIComponent(el2.innerText);
    document.write(s);
    el1.innerHTML = el2.innerText;
    document.location = el3.innerText;
  }
  bar();
}
foo();
function baz(a, b) {
 a.f = document.URL;
 document.write(b.f);
}
var x = new Object();
baz(x, x);
```

Rules

- A rule is a triple <Sources, Sinks, Sanitizers>
- Not all sources are valid for all sinks, and not all sanitizers are valid for all sinks

Rules

- A rule is a triple <Sources, Sinks, Sanitizers>
- Not all sources are valid for all sinks, and not all sanitizers are valid for all sinks

- Sources
  - Seeds of untrusted data
  - Field gets or returns of function calls
  - Ex: `document.url`

Rules

- A rule is a triple <Sources, Sinks, Sanitizers>
- Not all sources are valid for all sinks, and not all sanitizers are valid for all sinks

- Sources
  - Seeds of untrusted data
  - Field gets or returns of function calls
  - Ex: `document.url`
- Sinks
  - Security critical operations
  - Field puts or parameters to function calls
  - Ex: `element.innerHTML`

Rules

- A rule is a triple <Sources, Sinks, Sanitizers>
- Not all sources are valid for all sinks, and not all sanitizers are valid for all sinks

- Sources
  - Seeds of untrusted data
  - Field gets or returns of function calls
  - Ex: `document.url`
- Sinks
  - Security critical operations
  - Field puts or parameters to function calls
  - Ex: `element.innerHTML`
- Sanitizers
  - Marks flow as non-dangerous
  - Function calls
  - Ex: `encodeURIComponent(str)`

Complexities of JavaScript

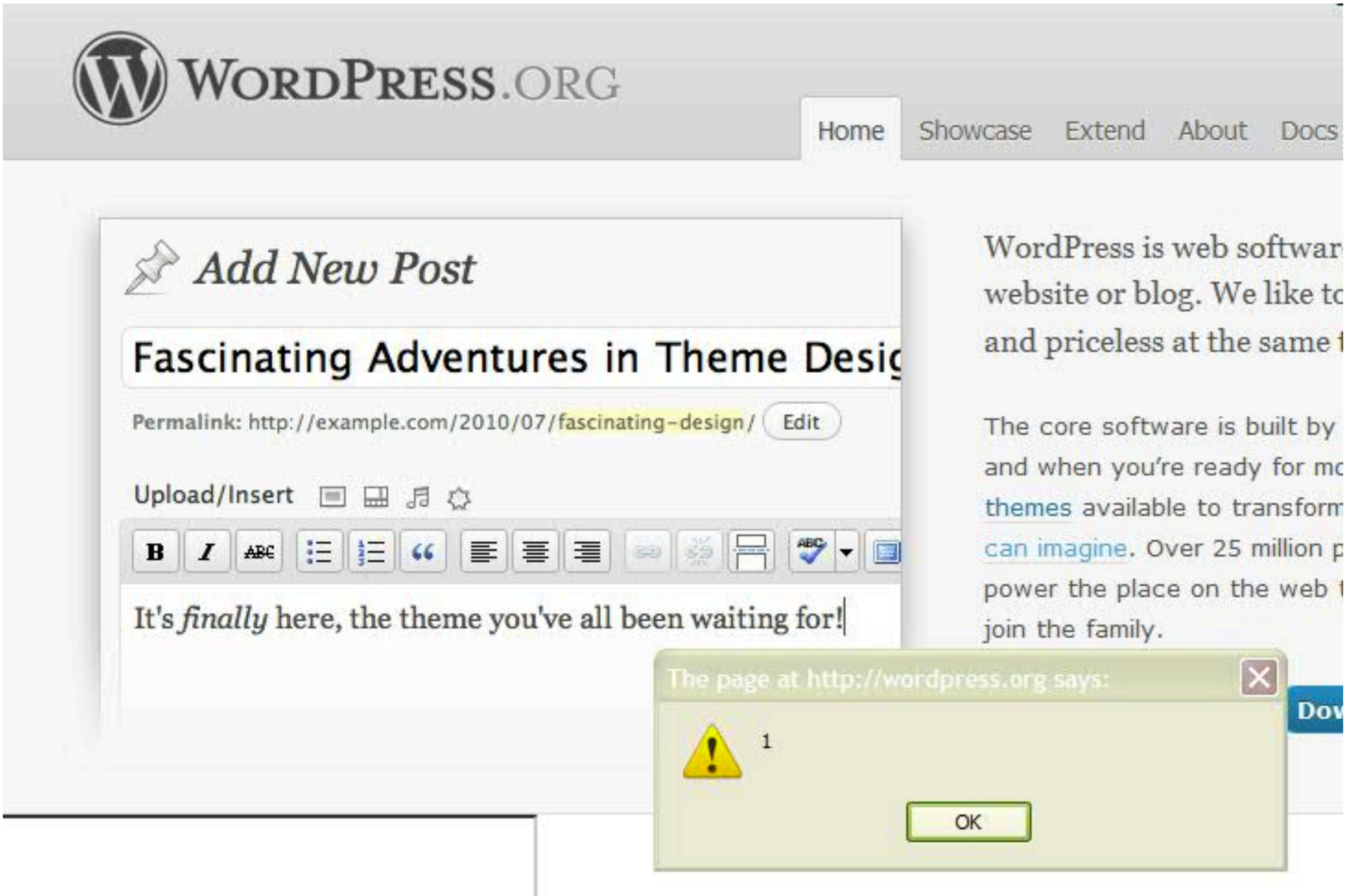
- Reflective property access
- Prototype chain property lookup
- Lexical scoping
- Function pointers
- `eval` and its relatives

```
eval("document.write('evil')");
```

Example

```
function foo(p1, p2) {
  p1.f = p2.f;
}

var a = new Object();
var b = new Object();
b.f = window.location.toString();

var c = new Object();
var d = new Object();
d.f = "safe";

foo(a, b);
foo(c, d);

document.write(a.f); // This is a taint violation
document.write(c.f); // This is NOT a taint violation
```

Since `d.f` is not tainted, `c.f` will not be tainted


WordPress.org
Home
Showcase
Extend
About
Docs
Add New Post
Fascinating Adventures in Theme Desig
Permalink: http://example.com/2010/07/fascinating-design/
Edit
Upload/Insert
It's finally here, the theme you've all been waiting for!
WordPress is web softwar
website or blog. We like to
and priceless at the same t
The core software is built by
and when you're ready for mo
themes available to transform
can imagine. Over 25 million p
power the place on the web t
join the family.
The page at http://wordpress.org says:
1
OK
Dow

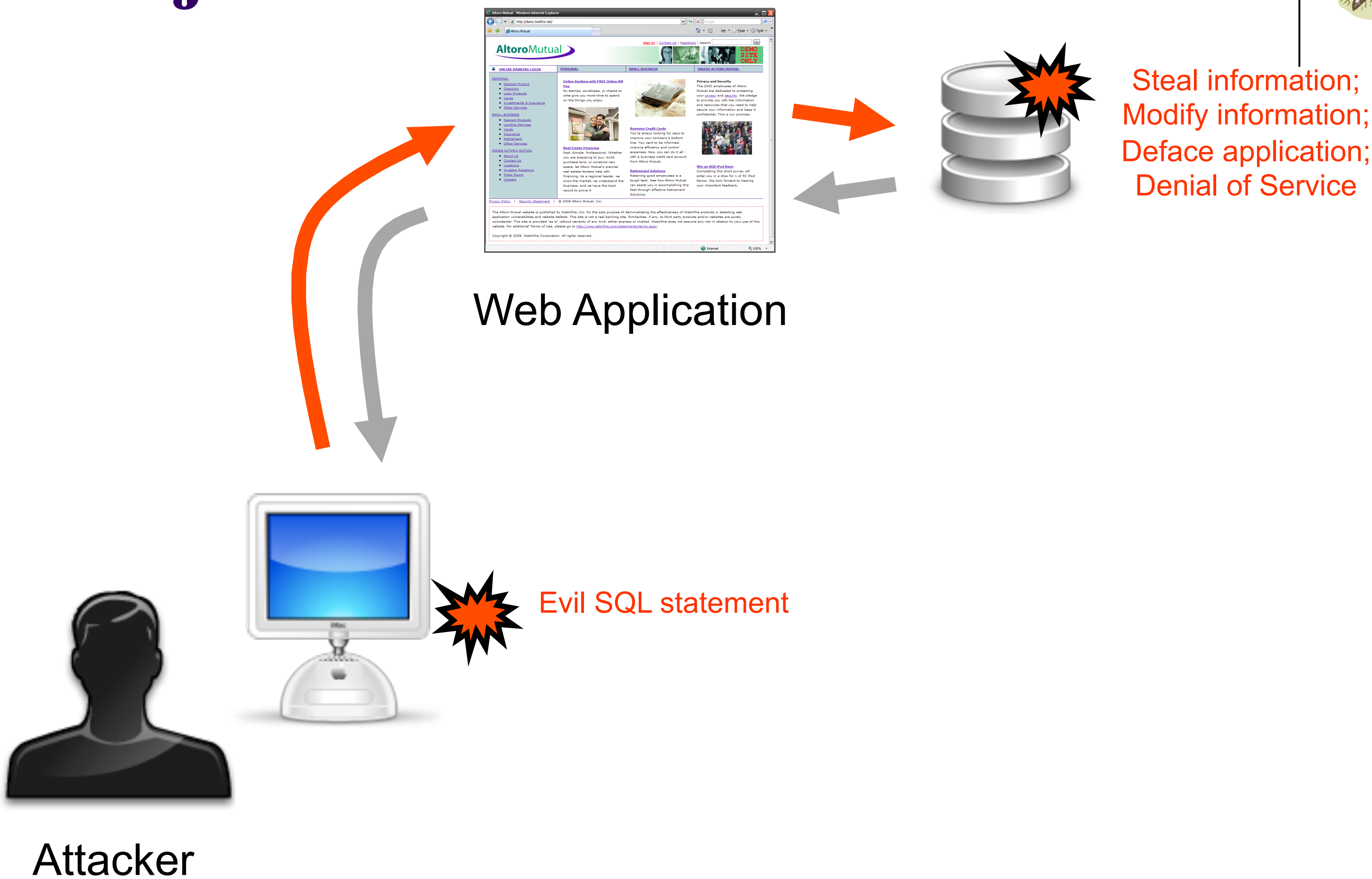
Part III

PHP Security

Security Support

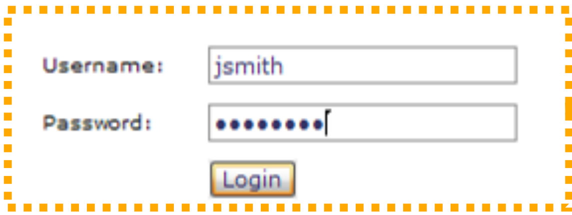
- Security APIs
  - Encryption
  - SSL
  - SSH
- Necessary to validate user input
  - Metacharacters
    - $ & ' " …
  - Wrong type of input
    - Dates
    - Numerical values
  - Too much input
    - HTML text areas can contain up to 8 MB

SQL Injection



SQL Injection in Code

```
String query = “SELECT * FROM users WHERE name=‘” +
   userName + “’ AND pwd=‘” + pwd + “’”;
```



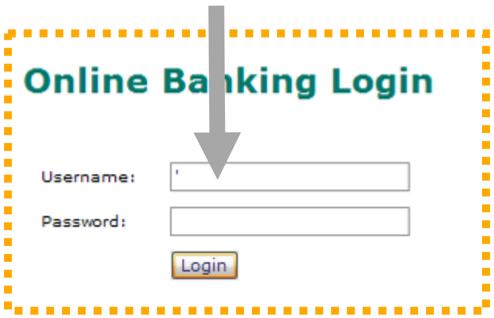

```
SELECT * FROM users WHERE name='jsmith' AND pwd='Demo1234'
```

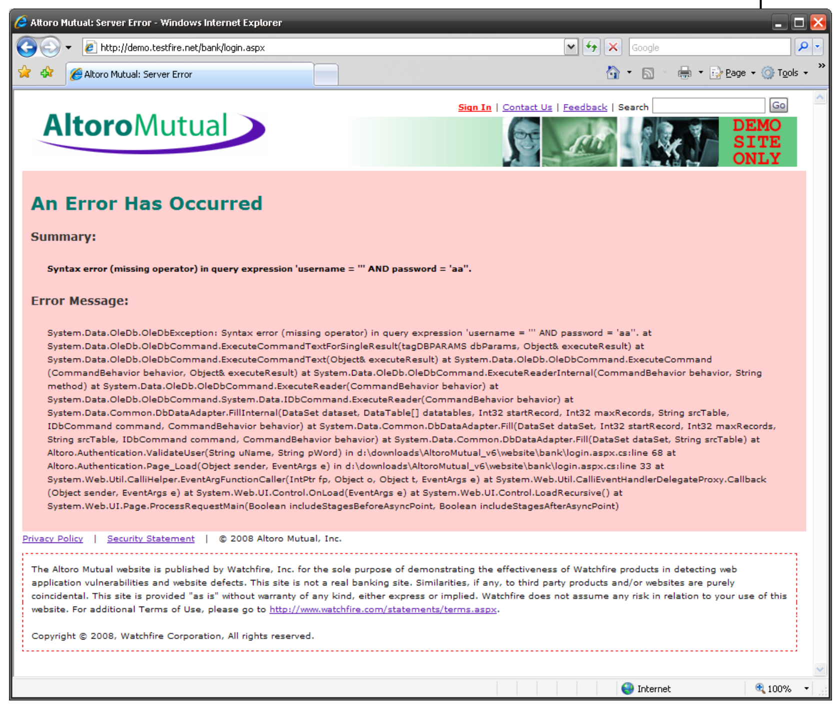
Ouch!

```
SELECT * FROM users WHERE name='foo';drop table custid;--' AND pwd=''
```

Checking for SQL Injection

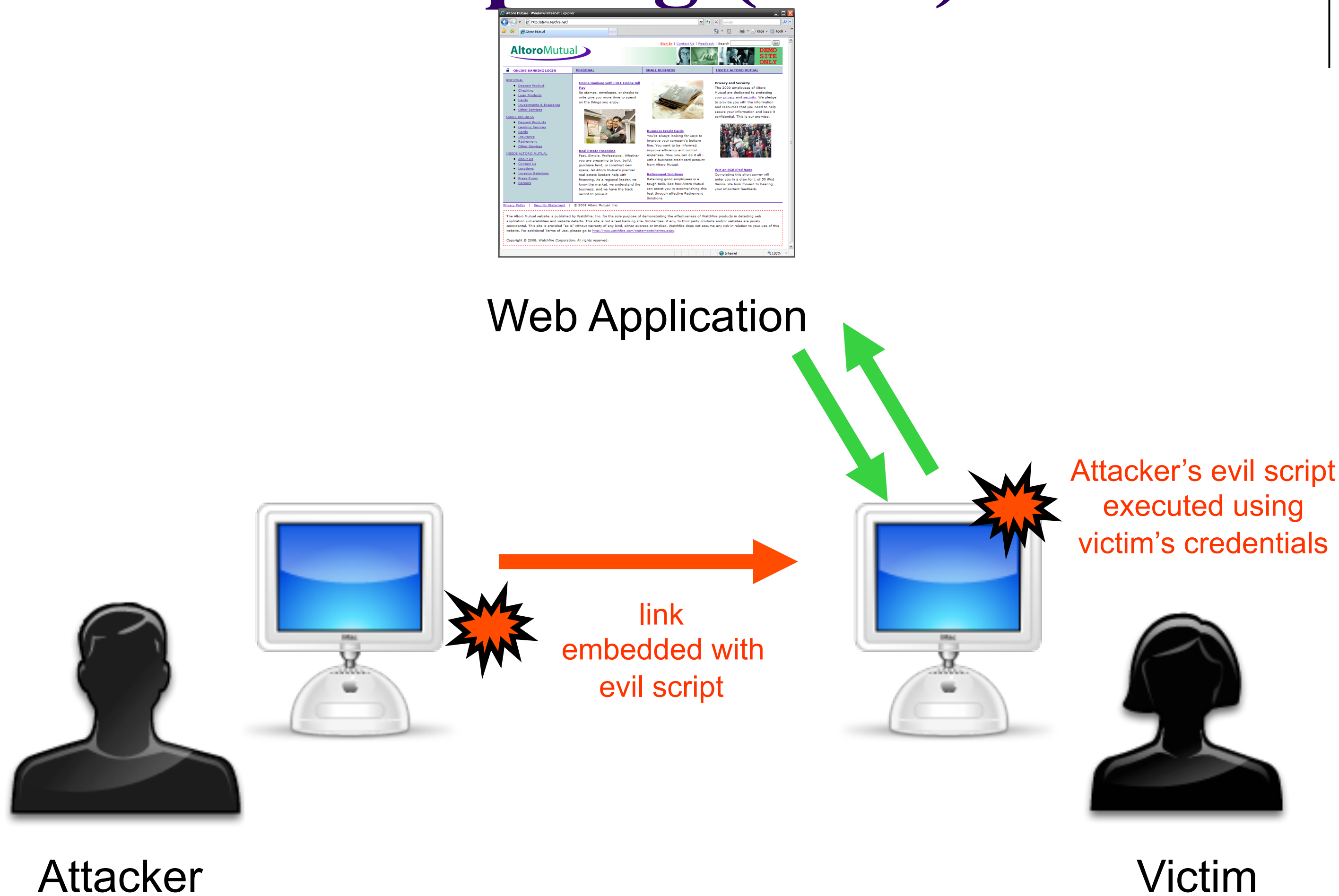
Put apostrophe into textbox



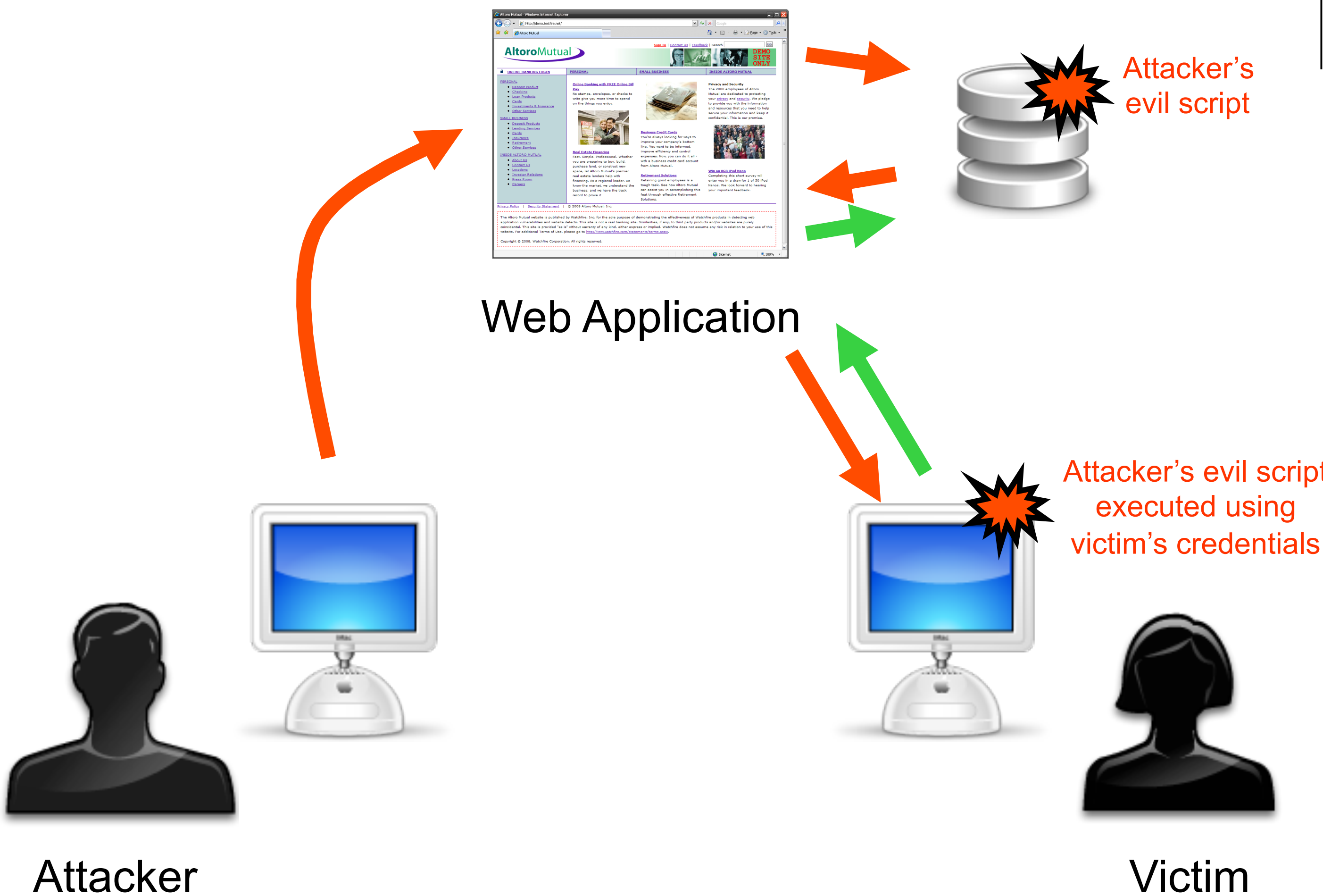


- Application responds with SQL error, suggesting to the attacker that string is being used to construct SQL query

Cross Site Scripting (XSS)



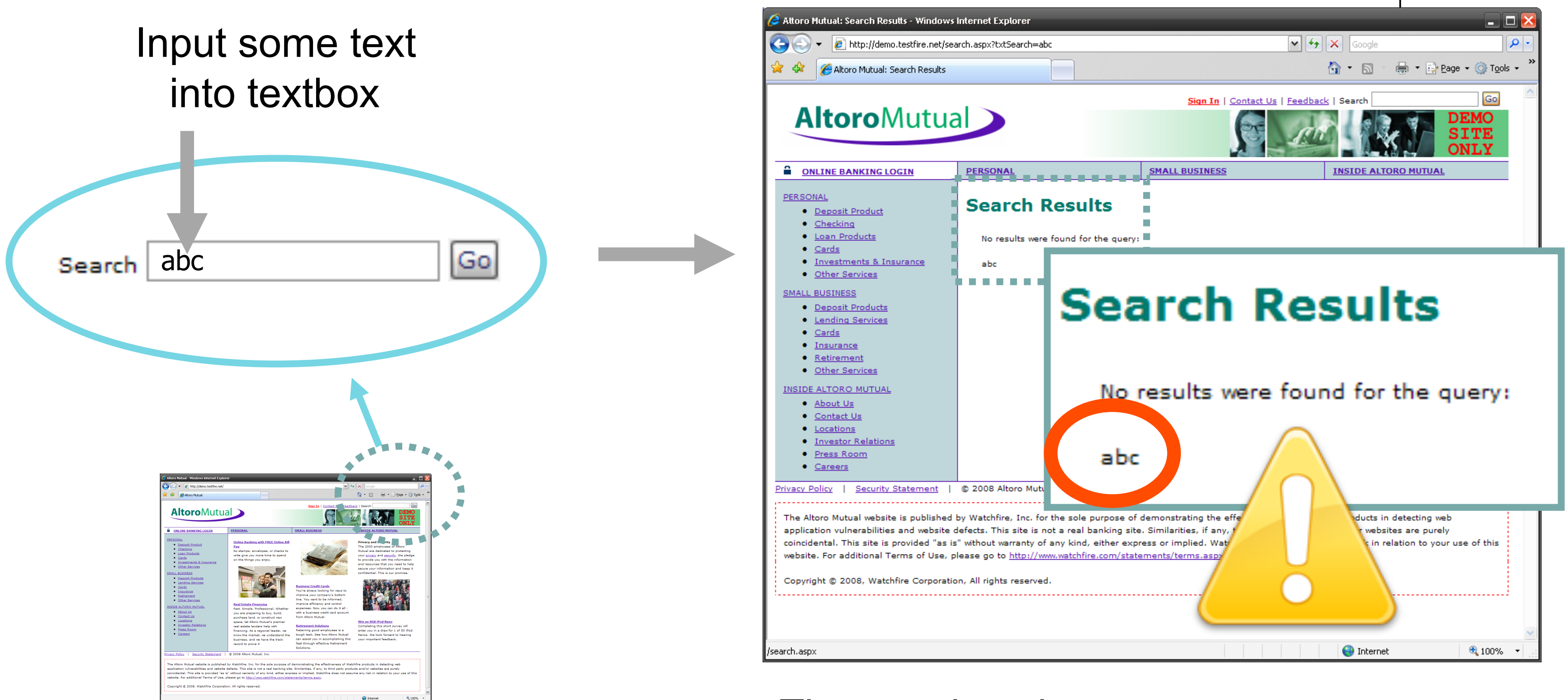
Stored XSS



AltoroMutual
Attacker's
evil script
Web Application
Attacker's evil script
executed using
victim's credentials
Attacker
Victim

Checking for XSS

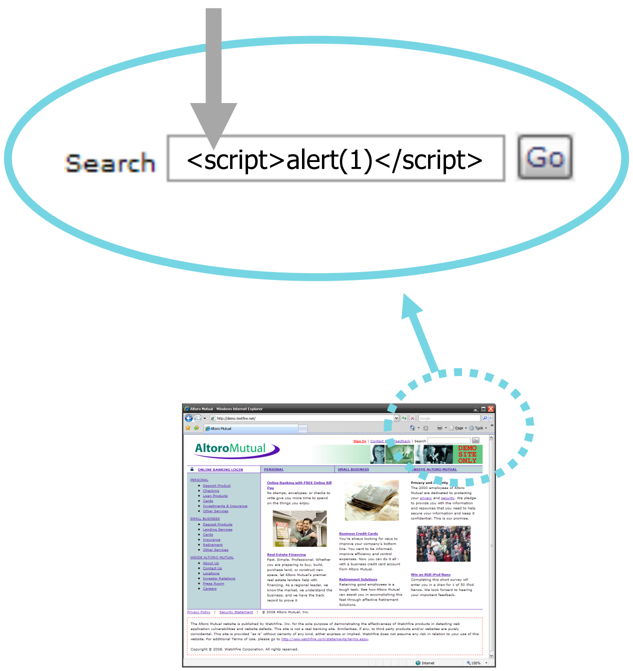
Input some text into textbox

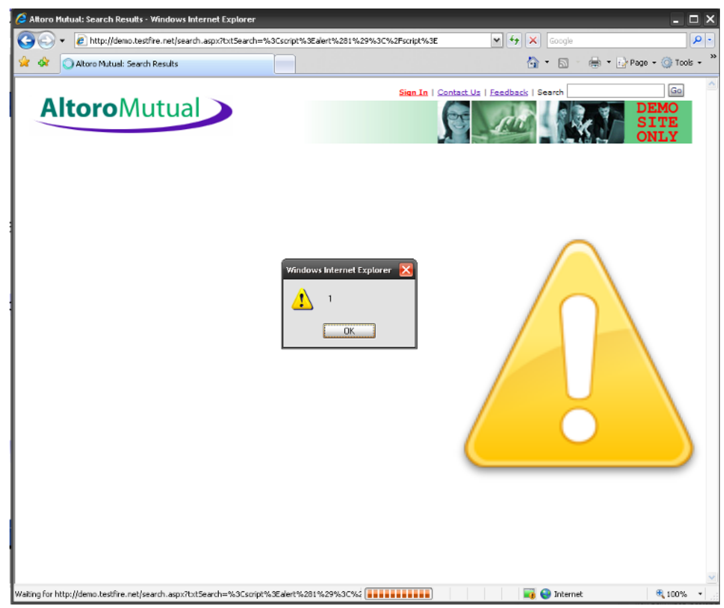


- The warning sign:
  User input embedded in HTML response

Checking for XSS (cont.)

Put an evil JavaScript
into the textbox





- Evil script was executed by browser
- Cause: Application did not apply HTML encoding
- Link containing this script could be sent to victim

What Needs to Be Validated?

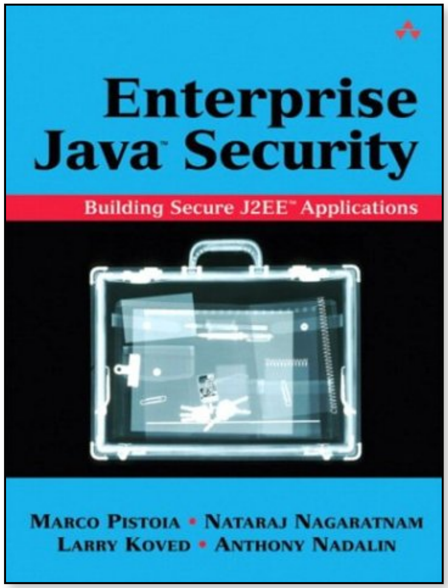
- ANY and ALL user input
- But also data coming from:
  - Database
  - Network
  - Application settings
  - Web services
  - File system
  - Command line arguments
  - Environment variables
- *Anything external to your application*

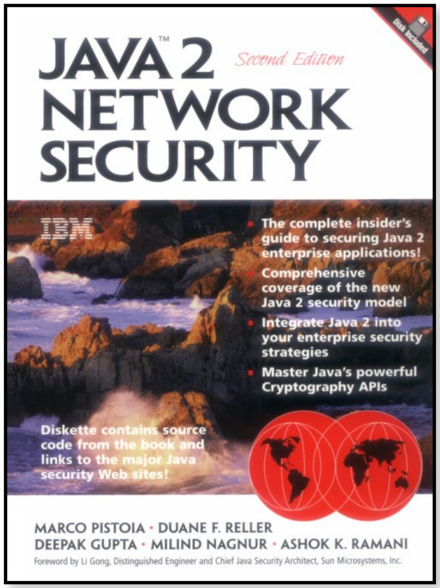
How to Use User Input and Stay Safe

- User input flows into **HTML page**? ✓ Apply **HTML encoding**!
- User input flows into **SQL command**? ✓ Apply **SQL encoding**!
- User input flows into **URL** or **HTTP Header**? ✓ Apply **URL encoding**!
- User input flows into **Log file**? ✓ Remove/encode **CR/LFs**!
- User input flows into a **command execution**? ✓ Apply **white-listing**!

Bibliography

- Salvatore Guarnieri, Marco Pistoia, Omer Tripp, Julian Dolby, Stephen Teilhet, Ryan Berg. *Saving the World Wide Web for Vulnerable JavaScript*. In Proceedings of the ISSTA 2011 Conference, Toronto, ON, Canada, July 2011.
- Paolina Centonze, Robert J. Flynn, and Marco Pistoia. *Combining Static and Dynamic Analysis for Automatic Identification of Precise Access-Control Policies*. In Proceedings of the Annual Computer Security Applications Conference (ACSAC 2007), Miami Beach, FL, December 2007.
- Marco Pistoia, Anindya Banerjee, and David Naumann. *Beyond Stack Inspection: A Unified Access-Control and Information-Flow Security Model*. In Proceedings of the **IEEE Symposium on Security and Privacy 2007**, Oakland, CA, May 2007.
- Marco Pistoia, Stephen J. Fink, Robert J. Flynn, and Eran Yahav. *When Role Models Have Flaws: Static Validation of Enterprise Security Policies*. In Proceedings of the 29th International Conference on Software Engineering (ICSE 2007), Minneapolis, MN, May 2007.
- Marco Pistoia, Satish Chandra, Stephen Fink, and Eran Yahav. *A Survey of Static Analysis Methods for Identifying Security Vulnerabilities in Software Systems*. **IBM Systems Journal**, volume 46, number 2, Armonk, NY, USA, May 2007. International Business Machines Corporation.
- Marco Pistoia and Francesco Logozzo. *Program Analysis for Security and Privacy*. In *Object-Oriented Technology: ECOOP 2006 Workshop Reader, Final Reports*. Twentieth European Conference on Object-Oriented Programming (ECOOP 2006), Nantes, France, July 2006. Lecture Notes in Computer Science (LNCS), volume 4379. Springer-Verlag.
- Paolina Centonze, Gleb Naumovich, Stephen J. Fink, and Marco Pistoia. *Role-Based Access Control Consistency Validation*. In Proceedings of the ACM SIGSOFT 2006 International Symposium on Software Testing and Analysis (ISSTA 2006), Portland, ME, USA, July 2006. ACM Press.
- Xiaolan Zhang, Larry Koved, Marco Pistoia, Sam Weber, Trent Jaeger, Guillaume Marceau, and Liangzhao Zeng. *The Case for Analysis Preserving Language Transformation*. In Proceedings of the ACM SIGSOFT 2006 International Symposium on Software Testing and Analysis (ISSTA 2006), Portland, ME, USA, July 2006. ACM Press.
- Marco Pistoia, Robert J. Flynn, Larry Koved, and Vugranam C. Sreedhar. *Interprocedural Analysis for Privileged Code Placement and Tainted Variable Detection*. In Proceedings of the 19th European Conference on Object-Oriented Programming (ECOOP 2005), pages 362-386, Glasgow, Scotland, UK, July 2005. Springer-Verlag.
- Larry Koved, Marco Pistoia, and Aaron Kershenbaum. *Access Rights Analysis for Java*. In Proceedings of the 17th ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA 2002), pages 359-372, Seattle, WA, USA, November 2002. ACM Press.





Questions?

pistoia@us.ibm.com
www.research.ibm.com/people/p/pistoia