Security for Web Languages

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

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

Top-Ten Web-Application Security Vulnerabilities

- Top-ten security vulnerabilities according to the Open Web Application Security Project (OWASP) (http://www.owasp.org)
- Current focus on analysis tools for:
  - Access control and secure configuration (7, 10)
  - Information flow (integrity and confidentiality) (1, 2, 3, 4, 5, 6)
  - Correct usage of security and cryptography APIs (9, 8)

Threat Evolution

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
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

<table>
<thead>
<tr>
<th>Principal</th>
<th>Request</th>
<th>Object</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>Admin</td>
<td>Read</td>
<td>File</td>
<td>True</td>
</tr>
<tr>
<td>Admin</td>
<td>Write</td>
<td>File</td>
<td>True</td>
</tr>
<tr>
<td>User</td>
<td>Read</td>
<td>File</td>
<td>False</td>
</tr>
<tr>
<td>User</td>
<td>Write</td>
<td>File</td>
<td>False</td>
</tr>
</tbody>
</table>

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 Schroeder, 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

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

Example: Injection Flaws in Web Applications

```java
public void submitQuery(String userName) {
    String query = 
```
```
    public void submitQuery(String userName) {
        String query = 
```
```
        `foo'; drop table custid;`
Why You Need to Know about Class Loaders

- If you are an application developer, you can take advantage of the security features of the class loading system.
- If you are a JVM or container developer, you need to know how to develop a secure class loader.

The Primordial Class Loader

- Not a Java class
- Written in native code
- Part of the JVM
- Responsible for loading system classes (specified in the boot class path)

Other Class Loaders

- Subclasses of the abstract class java.lang.ClassLoader
  - Available implementations:
    - java.security.SecureClassLoader subclasses ClassLoader
    - java.net.URLClassLoader subclasses SecureClassLoader
- Responsible for loading non-system classes
- At JVM startup:
  - ExtClassLoader is instantiated to load extension classes (specified in the extension class path)
  - AppClassLoader is instantiated to load local or remote application classes (specified in the application class path)
- Custom class loaders can be instantiated by the application with permissions

Name Spaces

- A class loader’s name space is the set of all classes that have been loaded by that class loader.
- Within a single name space, duplicated class names are prohibited.
- From a JVM perspective, the fully qualified name of a class includes the name of the class loader that loaded that class.

The Class Loading Delegation Tree

- Each class loader $B$, except the primordial, is a Java class.
- It itself is loaded by a class loader, $A$.
- Unless differently specified when $B$ was constructed, $A$ is $B$’s parent.
- This relationship generates a class loading tree.
- The primordial class loader is the root of the tree.

Correct Search Order for New Classes

- When a class triggers an invocation for a class that has not been loaded yet, its class loader is involved.
- Every class loader upon invocation delegates the request to its parent.
- This causes a call back all the way to the primordial class loader.
- The primordial class loader attempts to load the class from the boot class path.
- If it fails, it returns control to the extension class loader, etc.
- Only if all ancestors fail will the child try to locate and load the class.
Isolation

- Classes loaded by class loaders in different branches of the tree are isolated:
  - They cannot directly reference each other
  - Direct cross name visibility is forbidden
- Classes loaded from different remote locations are isolated

The Second Leg of Java Security: The Class File Verifier

- It is an integral part of the JVM, written in native code
- It makes sure that loaded classes cannot:
  - Contain illegal bytecode instructions
  - Forge pointers to protected memory
  - Over/underflow the program stack
  - Corrupt the JVM integrity
- It is there to prevent:
  - Java-compiler and Java-bytecode attacks
  - Non-Java-compiler attacks
  - Release-to-release binary compatibility problems
- It runs only on non-system classes

Core Java Packages Boundary Protection

- When a Java class is being loaded, and the class declares to belong to a trusted package, the class loader asks the security manager to check whether or not the code is authorized to define a class in that package
- This protects core packages from malicious code

Java-Compiler and Java-Bytecode Attacks

- All valid Java source code programs can be compiled to bytecode programs
  - A certified Java compiler only generates legal bytecode
  - A malicious Java compiler may generate programs attempting to subvert the integrity of the JVM
- There are bytecode programs that have no corresponding Java source
  - It is possible to generate Java bytecode programs from other high-level languages (such as NetRexx, COBOL)
  - It is possible to edit the bytecode

The Third Leg of Java Security: The Security Manager

- Enforces access-control restrictions
- It is an instance of java.lang.SecurityManager or one of its subclasses
  - The default security manager is java.lang.SecurityManager
  - Application developers can subclass the default security manager
- By default, applications run without a security manager
- A security manager can be enforced:
  - Statically, with the -Djava.security.manager command-line flag
  - Programmatically, by creating a new SecurityManager instance and passing it to java.lang.System.setSecurityManager()
  - java.lang.RuntimePermission "setSecurityManager"
  - java.lang.RuntimePermission "defineSecurityManager"
- There cannot be more than one security manager in effect at any time
  - The current security manager can be retrieved by calling System.getSecurityManager()
Types of Attack
- Attacks prevented by the default security manager:
  - System-modification attacks
    - A program gets write access and makes changes to the system
  - Privacy-invasion attacks
    - A program gets read access and steals private information
- Attacks the default security manager does not currently prevent:
  - Denial of service attacks
    - A program inappropriately uses system resources without authorization (CPU, AWT windows, etc.)

Security Manager’s Areas of Control
- The Security Manager class offers a number of check methods to enforce access control
  - checkRead(), checkWrite(), checkConnect(), etc.
- Each check method instantiates a java.security.Permission object representing the type of access being attempted and passes it to checkPermission()

The Java Access Control Model
- Java offers a fine-grained access-control model
- By default, the security manager denies access to all protected system resources
- It is possible to specify in detail what code is allowed to do based on:
  - The URL location from which the code is coming
  - The signers of the code
  - Multiple signers are allowed
  - The JVM verifies signatures transparently
  - The identity of the user running the code (with JAAS)
- Permissions are granted declaratively in a policy database

The Concept of Permission in Java 2
- A permission is the right to access a protected resource
- The Java 2 platform offers the abstract java.security.Permission class to represent permissions
- Each permission is represented as an object of a subclass of Permission
- Standard Permission classes are provided to represent the right to access protected system resources
- In Java, permissions are positive

The Permission Class Hierarchy
- A Permission object may have a target and an optional list of actions
  - The target is the object of the permission
  - The action is the type of access being attempted
The Permission implies( ) method

- The Permission class has an abstract instance method, implies()
- implies() takes a Permission object parameter and returns a boolean
- Given p and q Permission objects
  - If p.implies(q) returns true, then p is stronger or equivalent to q (granting p implicitly grants q too)
    - p = new FilePermission("C:/tmp/*", "read");
    - q = new FilePermission("C:/tmp/file.txt", "read");
  - If p.implies(q) returns false, then p is not stronger or equivalent to q, but this does not necessarily mean that q is stronger than p
    - p = new FilePermission("C:/tmp/*", "read");
    - q = new SocketPermission("www.ibm.com", "read");

Custom Permissions

- Sometimes, it is necessary to implement custom Permission subclasses to represent access to resources that are not protected by the standard Permission subclasses
  - For example, to access the ROM memory
  - java.security.BasicPermission is a concrete implementation of Permission useful if the Permission being implemented requires only a target
  - Non-standard Permission classes should be signed

Security Policy

- Permissions are granted to code declaratively in a policy database
- The J2SE reference implementation comes with a flat-file-based policy database, called the policy file
  - Editable with a text editor or with the Policy Tool (policytool)
  - Policy providers can supply more sophisticated implementations by subclassing the abstract class java.security.Policy
- There can be multiple policy files in effect at any time, listed in the $JAVA_HOME/jre/lib/security/java.security configuration file:
  - A user-defined policy file, $USER_HOME/.java.policy
  - A system-wide policy file, $JAVA_HOME/jre/lib/security/java.policy
  - A corporate-wide policy file, retrievable remotely
  - There is only one Policy object in effect at any given time
  - Policy files are portable across different platforms
  - Variables $JAVA_HOME, $USER_HOME, $FILE_SEPARATOR, etc. are resolved at runtime based on the underlying platform

Example 1 of Policy File grant Entry

```java
grant signedBy "ibm, tivoli", codeBase "http://www.ibm.com" {
    permission java.net.SocketPermission "www.ibm.com:80", "connect";
    permission java.io.FilePermission "C:/log.txt", "write";
}
```

Signed by

IBM

URL location

http://www.ibm.com

Example 2 of Policy File grant Entry

```java
grant signedBy "ibm", codeBase "http://www.ibm.com" {
    permission java.net.SocketPermission "www.ibm.com:80", "connect";
    permission java.io.FilePermission "C:/log.txt", "write";
}
```

Signed by

IBM

URL location

http://www.ibm.com

Stack-Based Access Control Systems

- In a SBAC system, when a restricted resource is being accessed, all the callers on the stack must exhibit the permission to access that resource
- A stack walk is performed by an access-control function
  - Permissions
    - Can be granted to both code and users
    - Are assigned by the class loaders
    - Are granted in policy databases
  - Examples of SBAC systems:
    - Java 2
    - Microsoft .NET Common Language Runtime (CLR)
Permissions Granted to a Thread of Execution

- Each of the classes on the thread stack must have been granted at least one permission that implies the permission being checked.
- If even one of the classes on the thread stack has not been granted a permission that implies the permission being checked, then the operation fails.
- The set of permissions effectively granted to a thread of execution is the intersection of the sets of permissions implied by the single protection domains on the stack.

When a New Thread is Created

- When an application creates a child thread, a snapshot from the spawning thread is used to set the top-of-stack protection domains.
- This prevents the child thread from performing operations that the spawning thread could not legally perform itself.

Same Code, Multiple Stacks

- Need for Privilege-Asserting Code

- Multiple Permission Requirements

SBAC with Privilege-Asserting Code
Proof that programs accepted by IBAC are noninterferent

IBAC is based on the idea that any access-control policy implicitly defines an information-flow policy [Denning & Denning 1977]. The access-control policy assigns sets of permissions to program components. The access-control policy assigns sets of permissions to program components.

For any security-sensitive operation requiring permission \( p \), IBAC imposes the integrity property that no code component without permission \( p \) be allowed to influence the execution of that operation.

Proof that programs accepted by IBAC are noninterferent.

Problems with SBAC

Authorization decisions are based on the entire history of code that was ever executed [Fournet & Gordon: POPL 2002, Abadi & Fournet: NDSS 2003]. For any security-sensitive operation guarded by a primitive:

- IBAC imposes the integrity property that no code component without permission \( p \) be allowed to influence the execution of that operation.
- Proof that programs accepted by IBAC are noninterferent.

Problems with HBAC

HBAC is safe, but potentially too restrictive. HBAC may unjustly prevent legitimate programs from executing. HBAC may behave inconsistently. HBAC does not prevent tainted variables from influencing the execution of privilege-asserting code.

Side effects

HBAC may unjustly prevent legitimate programs from executing.

HBAC may behave inconsistently.

HBAC does not prevent tainted variables from influencing the execution of privilege-asserting code.

Information-Based Access Control

We introduce and formalize IBAC: a new access-control model that, for any security-sensitive operation, verifies that all the code responsible for that operation is sufficiently authorized [K&P97].

- More restrictive (and precise) than SBAC.
- Less restrictive (and precise) than HBAC.
- IBAC is based on the idea that any access-control policy implicitly defines an information-flow policy [Denning & Denning 1977].
- The access-control policy assigns sets of permissions to program components.
- Information-flow labels automatically derived from access-control labels, which can be automatically computed [KOPSILV92, ICSE97].
- No need to insert information-flow check points.
- For any security-sensitive operation requiring permission \( p \), IBAC imposes the integrity property that no code component without permission \( p \) be allowed to influence the execution of that operation.
- Proof that programs accepted by IBAC are noninterferent.

Problems with IBAC

HBAC attaches dynamic labels on values and on the program counter.

Each value’s label is the set of permissions of the principal owning the code that created that value.

HBAC augments dynamic stack inspection with tracking of information flow.

A call to the test primitive:
- Checks all the callers up to the first grant call.
- Verifies all the labels on the values read in the security-sensitive operation guarded by test.

IBAC Integrity Principle

TB is the set of values that may be read by a security-sensitive operation.

Inconsistent behavior.

HBAC may not prevent tainted variables from influencing the execution of privilege-asserting code.
Role-Based Access Control Systems

- A role is a set of permissions that can be granted to users and/or groups of a computer system.
- Each permission represents the right to perform a security-sensitive operation; it does not directly represent the right to access security-sensitive data or resources.
- Examples of RBAC systems:
  - Java, Enterprise Edition (Java EE)
  - Microsoft .NET Common Language Runtime (CLR)

Role Definition in Java EE

- Roles are application-specific.
- They are defined in the deployment descriptors of an application’s components.
- They can be used to restrict access to enterprise methods.

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

- Similar to Java.
- Scripts are confined inside a sandbox where they cannot have access to the operating 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

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, &lt;h&gt;John&lt;/h&gt;!
- 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, &lt;b&gt;John&lt;/b&gt;&lt;script&gt;alert('Uh oh');&lt;/script&gt;&lt;/b&gt;
- 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

Part IV

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

- Attacker
- Web Application
- Evil SQL statement
- Steal information; Modify information; Deface application; Denial of Service

SQL Injection in Code

```java
String query = "SELECT * FROM users WHERE name='jsmith' AND pwd='Demo1234';"
```

- String query = "SELECT * FROM users WHERE name='" + userName + "' AND pwd='" + pwd + "';"

Ouch!

Checking for SQL Injection

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

Cross Site Scripting (XSS)

- Attacker Victim
- Web Application
- Link embedded with evil script
- Attacker's evil script executed using victim's credentials

Stored XSS

- Attacker Victim
- Web Application
- Attacker's evil script stored in database
- Victim's web application retrieves data from database
- Attacker's evil script embedded in retrieved data
- Victim executes attacker's evil script
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 CRLF!
- User input flows into a command execution? ✓ Apply white-listing!

Bibliography


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