Common Security Weaknesses in Java Web Apps and How to Avoid them

Joe Fisher, President
Affinity IT Security Services

Presented at
QCon NYC
June 18th, 2012

Topics

- A “Holistic” Approach
- Common Vulnerabilities
  - The OWASP Top 10 2010
  - The CWE Top 25 2011
- Application Security and the SDLC
- Summary
- Additional Resources
- A Vulnerability in Depth
- Thank You
- Q & A

A “Holistic” Approach

- Motivation for a Holistic Approach
  - You can’t “Test In” Security after development
    - You would never know when you are done
    - You could not plan the Testing interval
    - Discovered flaws might warrant redesign
  - Diligent attention needed throughout entire SDLC
    - Maintaining a focus on security produces secure products
    - Proper considerations at the proper time
    - Later Phases depend on deliverables from earlier phases

Common Vulnerabilities

- The OWASP Top 10 2010
  - A1: Injection
  - A2: Cross-Site Scripting
  - A3: Broken Authentication and Session Management
  - A4: Insecure Direct Object References
  - A5: Cross-Site Request Forgery
  - A6: Security Misconfiguration
  - A7: Insecure Cryptographic Storage
  - A8: Failure to Restrict URL Access
  - A9: Insufficient Transport Layer Protection
  - A10: Unvalidated Redirects and Forwards

Common Vulnerabilities

- The CWE Top 25 2011
  - CWE-89: SQL Injection
  - CWE-78: OS Command Injection
  - CWE-120: Classic Buffer Overflow
  - CWE-79: Cross-Site Scripting
  - CWE-306: Missing Authentication for Critical Function
  - CWE-862: Missing Authorization
  - CWE-798: Use of Hard-coded Credentials
  - CWE-311: Missing Encryption of Sensitive Data
  - CWE-434: Unrestricted Upload of File with Dangerous Type
  - CWE-807: Reliance on Untrusted Inputs in a Security Decision
  - CWE-250: Execution with Unnecessary Privileges
  - CWE-352: Cross-Site Request Forgery
  - : continued

Common Vulnerabilities

- The CWE Top 25 2011 (continued...)
  - CWE-22: Path Traversal
  - CWE-494: Download of Code without Integrity Check
  - CWE-863: Incorrect Authorization
  - CWE-829: Inclusion of Functionality from Untrusted Control Sphere
  - CWE-732: Incorrect Permission Assignment for Critical Resource
  - CWE-676: Use of Potentially Dangerous Function
  - CWE-327: Use of Broken or Risky Cryptographic Algorithm
  - CWE-131: Incorrect Calculation of Buffer Size
  - CWE-307: Improper Restriction of Excessive Authentication Attempts
  - CWE-601: URL Redirection to Untrusted Site
  - CWE-134: Uncontrolled Format String
  - CWE-190: Integer Overflow or Wraparound
  - CWE-759: Use of One-Way Hash without a Salt

Security and the SDLC

- Application Security and the SDLC
  - Project Initiation
  - Application Security Requirements
  - Designing Secure Software
    - Avoiding Common Vulnerabilities
  - Construction
    - Avoiding Common Vulnerabilities
  - System Testing
  - System Deployment

What can we do in each phase to improve product security ?

Security and the SDLC

- Project Initiation
  - Deliverable: Project Charter
    - Business Case
      - Estimated Project Cost
      - Estimating Project Benefits
      - Estimating Security Benefits
    - Quantifying Application Security Benefits
      - Reducing risk of negative consequences is beneficial

Application Security Costs Money

Quantified Damage Estimate = PoB * AD

- PoB: Probability of a Breach through application
- AD: Aggregate Damage --Worst Case Scenario

Security and the SDLC

- Aggregate Damages – Worst Case Scenario
  - Value of lost or damaged Assets
  - Damage to reputation or brand
  - Short term revenue loss
  - Emergency Response and associated Forensics
  - Remediation Costs
  - Costs associated with Preventing Future Attacks
  - Impact on Stock Price / Market Capitalization
  - Loss of IP or Competitive Advantage
  - Opportunity Costs

Security and the SDLC

- Requirements Gathering
  - Discovering and Documenting...
    - Applications Requirements
    - Application Security Requirements
  - Validating
    - Application Requirements Review
    - Application Security Requirements Review
      - Authentication
      - Authorization
      - What is Sensitive ?
      - Encryption
      - Infrastructure
      - :

Security and the SDLC

- Application Security Requirements
  - Form follows Function

  - You get what you specify
  - You can't "Test In" Security after development
  - Retrofitting Requirements is clumsy at best

Security and the SDLC

- Requirements Gathering
  - Applications Requirements
    - Assumptions and Constraints
    - Functional Requirements
    - User Interface Requirements
      - Windows/Pages, Content, Look and Feel
      - Navigation
    - Performance Requirements
    - System Interface Requirements
    - Data exchange
    - Client/Server Services
    - Infrastructure Requirements
    - Network Devices
    - Servers / Services

What the Application must do…

And the users exclaimed with a laugh and a taunt…

It's just what we asked for … but not what we want !

Security and the SDLC

- Requirements Gathering
  - Application Security Requirements
    - Meta-Data for all Inputs
    - Data Handling: Storage and Transport
    - Access Control: Authentication, Authorization
    - Session Management
    - Logging / Accountability
    - Error Handing
    - Remote Access
    - Protocols / Versions
    - System Upgrades
    - Infrastructure Requirements
    - System Monitoring

A6

Security and the SDLC

- Security Requirements Review
  - Objective
  - Participants
  - Deliverables
  - Remediation Process
- Security Requirements
  - Will guide Design and Development
  - Important input to Security Testing

It is good to have defined the role and authority of the Security Review Team in advance of any Review activity

Security and the SDLC

- Designing Secure Software
  - Finite State Model of Application
    - Navigation and Data Transfer
  - Authentication — A1, A3, CWE-306, CWE-307
  - Authorization — CWE-862, CWE-863, CWE-732, CWE-250
    - Enforcing Least Privilege
  - Data Management — CWE-807
    - Data Utilization
    - Input Validation Mechanism — A1, CWE-89, CWE-78, CWE-22
    - Data Sanitization — A2, CWE-78, CWE-89
  - State Management
    - Tracking Application State
    - Transition Validation — A2, A5, A8, A10, CWE-252, CWE-601
    - Session Management — A3

Security and the SDLC

- Designing Secure Software (... continued)
  - Data Protection — A7, A9, CWE-311, CWE-327, CWE-759
    - Data in Motion
    - Data at Rest
  - Secure System Updates — CWE-494
  - Avoiding Common Vulnerabilities
    - Direct Object References — A4

90% of OWASP-10 Potentially Addressed in Design

64% of CWE-25 Potentially Addressed in Design

SDLC Security Checkpoint

- Security Design Review
  - Objective
  - Participants
  - Deliverables
  - Remediation Process

It is good to have defined the role and authority of the Security Review Team in advance of any Review activity

YOU ESPECIALLY NEED SUBJECT MATTER EXPERTISE HERE …

Important Activity !

PAY NOW OR PAY LATER…

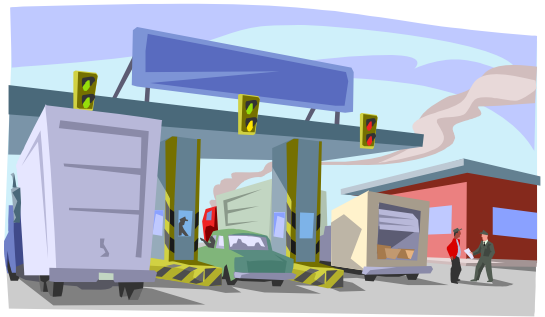
Security and the SDLC

- Construction
  - Validating Preconditions
  - Avoiding Common Vulnerabilities
    - Buffer Overflow ---------- CWE-120, CWE-131
    - Path Traversal ---------- A4, CWE-22
    - SQL Injection ---------- A1, CWE-89
    - Command Injection ---------- A2, CWE-78
    - Format Strings ---------- CWE-676, CWE-134
    - Integer Overflow / Wraparound ---------- CWE-190
    - :
  - Coding Standards, Guidelines, and Reviews
  - Managed vs. Unmanaged Code

SDLC Security Checkpoint

- Keys to Success are
  - Awareness
  - Compliance
- Security Code Review
  - Objective
  - Participants
  - Deliverables
  - Remediation Process

It is good to have defined the role and authority of the Security Review Team in advance of any Review activity



Security and the SDLC

- System Testing
  - Testing Application Requirements
    - Functional, User Interface, …
    - Requirement Specification Test Cases
  - Application Security Testing
    - Testing Security Requirements
      - Security Test Cases
      - Client/Server Input Validation
      - Testing for Information Leakage
      - Positive and Negative Test Cases
      - Error Handling
    - Internal Penetration Testing
      - Testing for Common Vulnerabilities
    - Pen Testing

Fitness for Use

Information Security

Security and the SDLC

- System Deployment
  - Synchronizing Environments:
    - Development, Test, Production
  - Database Configuration and Accounts
  - OS Configuration and Accounts
  - Password Management
  - Configuring & Testing Monitoring Resources
  - Unlinked Pages, Unused Features
  - Debug Code
  - Comments

Summary

- Take Aways...
  - You get what you specify,
    - Not necessarily what you want
    - Not necessarily what you need
  - You can't "Test In" Security after the fact
  - Security must be addressed throughout the entire SDLC
  - Security is not free and costs must be justified in Business Case
  - The most common vulnerabilities are well understood and largely preventable
  - Design Phase is especially critical to security
  - A focus on Organizational Change is necessary to change corporate culture

Additional Resources

- Open Web Application Security Project (OWASP)
  - www.owasp.org
- Common Weakness Enumeration (CWE)
  - cwe.mitre.org

A Vulnerability in Depth

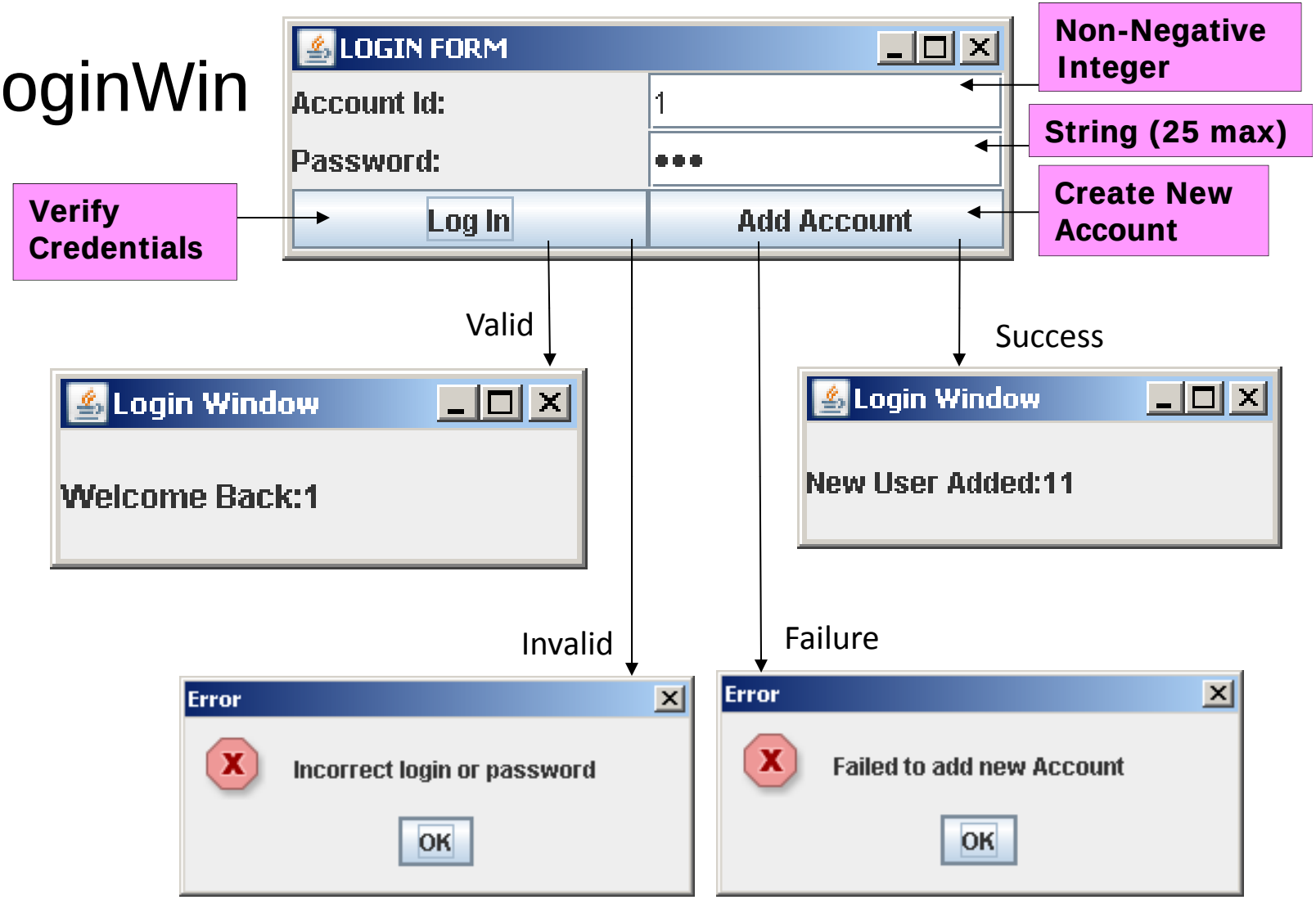
- What is SQL Injection ?
  - Attacker crafts malicious input that is executed by database
    - Input that perverts meaning of application SQL Statements
  - Resulting from failure to …
    - Utilize Parameterized Statements
    - Perform Input Validation
    - Pattern White-listing
    - Bounds Checking
    - Perform Data Sanitization

A Vulnerability in Depth

- Data Sanitization
  - Escaping characters and character sequences that have meaning to other components
    - Database (SQL)
    - Command (Shell, Perl, Python, etc.)
    - HTML
    - LDAP Query
    - :

A Vulnerability in Depth

- LoginWin



A Vulnerability in Depth

```
// Code omitted ...
    public void actionPerformed(ActionEvent ae) {
      String id = text1.getText();
      String pw = text2.getText();
      if (ae.getSource() == loginButton) {
        if (DBAccessor.verifyAccount (id, pw)) {
                // code omitted...
        }
        else{
          JOptionPane.showMessageDialog(this, "Incorrect login or password",
                                        "Error", JOptionPane.ERROR_MESSAGE);
        }
       }
       else {
         if (DBAccessor.addAccount (id, pw)) {
                // code omitted...
         }
         else {
           JOptionPane.showMessageDialog(this, "Failed to add new Account",
                                        "Error", JOptionPane.ERROR_MESSAGE);
         }
       }
    }   // Code omitted ...
```

LoginWin Class

Button Click Handler uses DBAccessor class to verify credentials and add accounts

A Vulnerability in Depth

```
// Code omitted ...
public static boolean verifyAccount (String loginId, String pw) {
    Connection conn = null;
    Statement stmt = null;
    ResultSet rs = null;
    try {
      // code omitted ...
      rs = openResultSet (stmt,
         "SELECT * FROM TopSecretAccounts WHERE ID=" +
         loginId + " AND HashedPW='" + hash(pw) + "'");

      return rs.next();
     }
     catch (Exception e) {
        System.err.println ( e.toString() );
     }
     finally {
         closeThingsDown (rs, stmt, conn);
     }
     return false;
}
// Code omitted ...
```

Password hashed

"true" if row found

DBAccessor Class

verifyAccount method returns true if credentials in accounts table

A Vulnerability in Depth

```
// Code omitted ...
public static boolean addAccount (String loginId, String pw) {
    Connection conn = null;
    Statement stmt = null;
    try {
      // code omitted …
      return stmt.executeUpdate (
                  "INSERT INTO TopSecretAccounts VALUES (" +
                  loginId + ", '" + hash(pw) + "')" ) == 1;
    }
    catch (Exception e) {
      System.err.println ( e.toString() );
    }
    finally {
      closeThingsDown (null, stmt, conn);
    }
    return false;
}
// Code omitted ...
```

"true" if row successfully inserted

Password hashed

DBAccessor Class

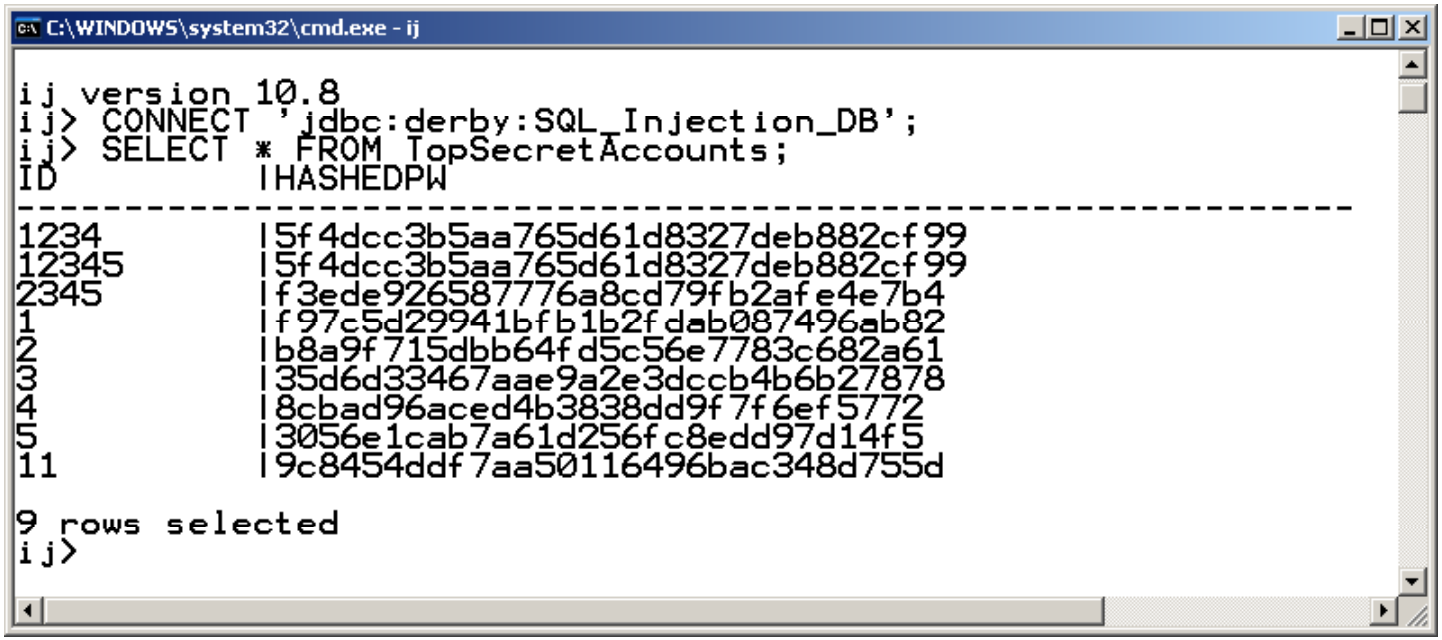
addAccount method returns true if credentials in accounts table

A Vulnerability in Depth

- Table Definition

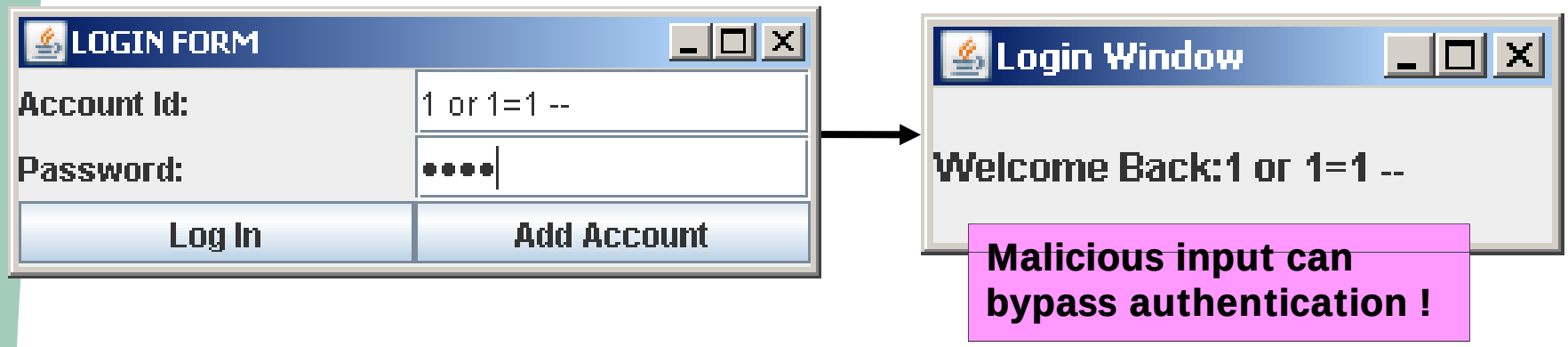
Example uses Java DB (Derby)

```
CREATE TABLE TopSecretAccounts (
   ID   INT    PRIMARY KEY,
   HashedPW   VARCHAR (50)
);
```



A Vulnerability in Depth

- Let's do some SQL Injection...



- Whoa.... Let's review the SQL again...

```
"SELECT * FROM TopSecretAccounts WHERE ID=" +
    loginId + " AND HashedPW='" + hash(pw) + "'");

"SELECT * FROM TopSecretAccounts WHERE ID=" +
    "1 or 1=1 --" + " AND HashedPW='" + “anything” + "'");
```

```
SELECT * FROM TopSecretAccounts WHERE ID= 1 or 1=1 -- AND
          HashedPW='anything';
```

-- denotes comment

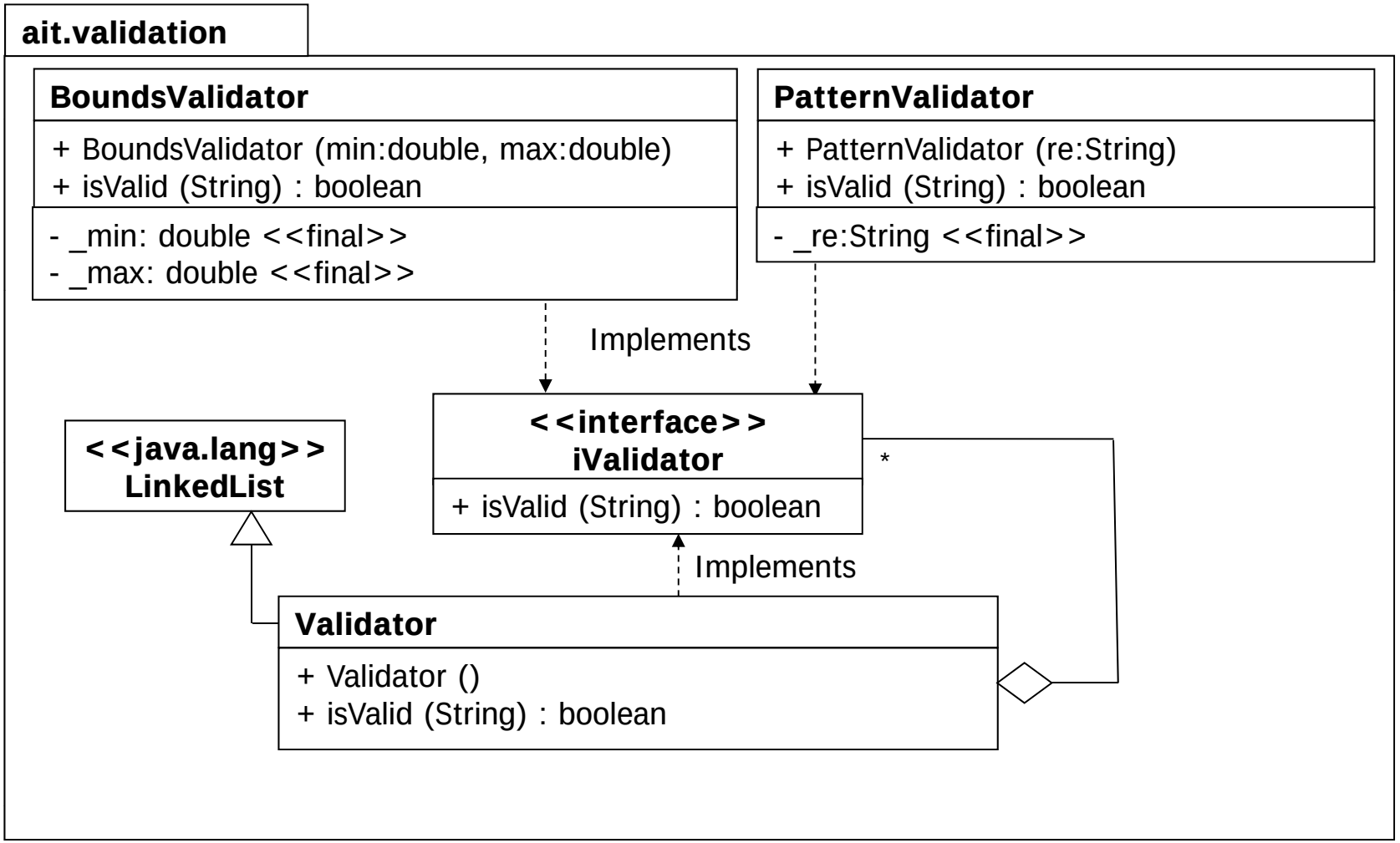
A Vulnerability in Depth

- Mitigation Options
  - Diligent Data Management
  - Parameterized Queries

```
// in SecureDBAccessor.verifyAccount
String pQuery =
  "SELECT * FROM TopSecretAccounts WHERE ID=? AND HashedPW=?";
PreparedStatement ps = conn.prepareStatement (pQuery);
ps.setInt (1, loginId); ps.setString (2, hash(pw));
ResultSet rs = ps.executeQuery ();
```

  - Design Reviews
  - Code Reviews
  - Security Testing
  - Error Handling
  - Least Privilege

A Vulnerability in Depth



A Vulnerability in Depth

- package ait.validation;
  - iValidator.java

```
public interface iValidator {
   public boolean isValid (String s);
 }
```

  - BoundsValidator.java

```
public class BoundsValidator implements iValidator {
   public BoundsValidator (double min, double max) {
         min = min;
        _max = max;
   }
   public boolean isValid (String s) {
        Double dObj = new Double (s);
        double dVal = dObj.doubleValue();
        return dVal >= _min && dVal <= _max;
   }
   private final double _min;
   private final double _max;
}
```

A Vulnerability in Depth

- package ait.validation;
  - PatternValidator.java

```
import java.util.regex.*;

public class PatternValidator implements iValidator {
   public PatternValidator (String re) {
      _re = Pattern.compile (re);;
   }
   public boolean isValid (String s) {
      return _re.matcher (s).matches();
   }
   private final Pattern _re;
 }
```

A Vulnerability in Depth

- package ait.validation;
  - LengthValidator.java

```
public class LengthValidator implements iValidator {
   public LengthValidator (long minLen, long maxLen) {
      _minLen = minLen;
      _maxLen = maxLen;
   }
   public boolean isValid (String s) {
      int sLen = s.length();
      return sLen >= _minLen && sLen <= _maxLen;
   }
   private final long _minLen;
   private final long _maxLen;
}
```

A Vulnerability in Depth

- package ait.validation;
  - Validator.java

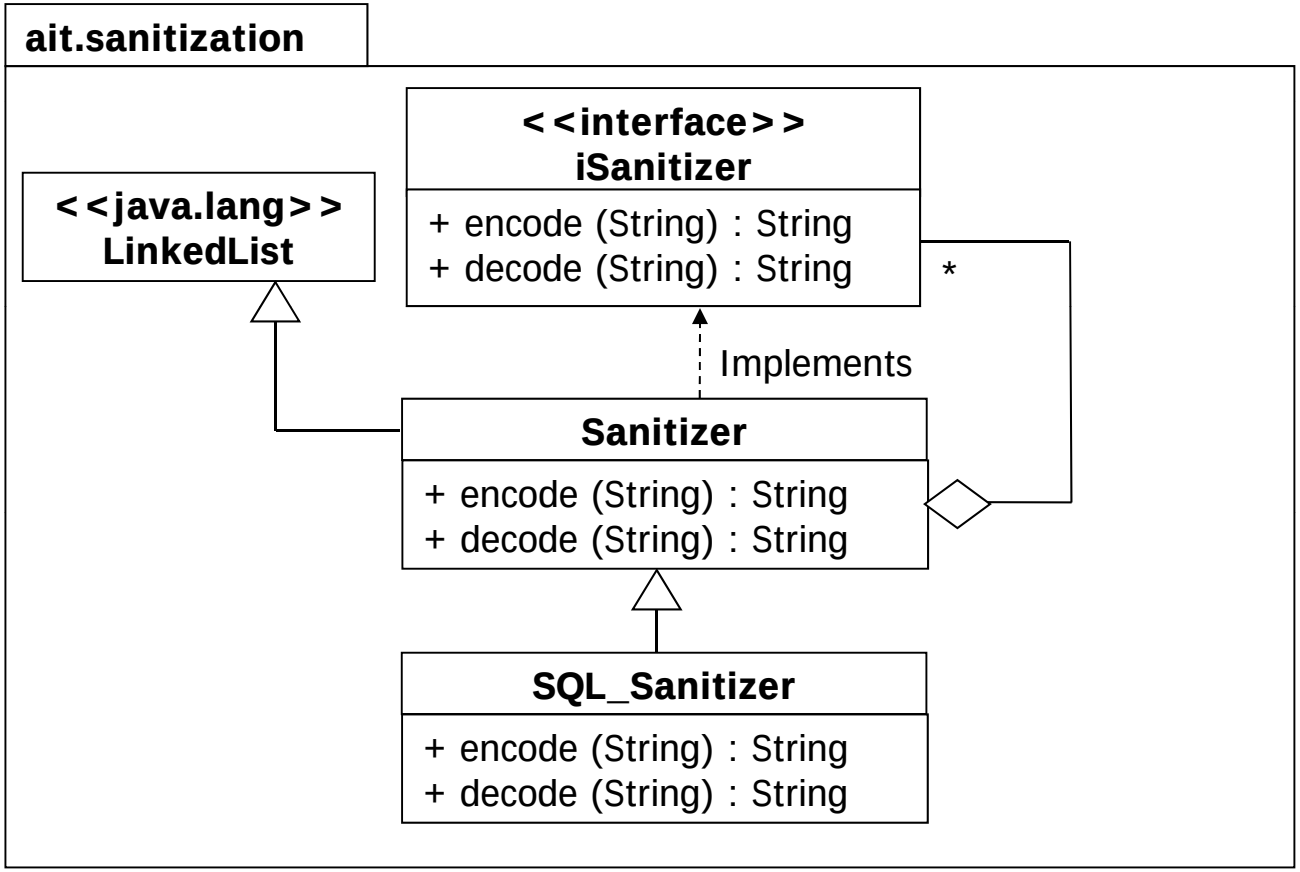
```
import java.util.*;

public class Validator extends LinkedList<iValidator>
                       implements iValidator {
   public Validator () {  }
   public boolean isValid (String s) {
    Iterator i = iterator();
    while (i.hasNext()) {
       try {
         iValidator v = (iValidator) i.next();
         if (! v.isValid(s)) return false;
       }
       catch (Exception e) {
         return false;
       }
    }
    return true;
   }
```

A Vulnerability in Depth

- SecureDBAccessor.java

```
// Code omitted ...  This appears right before SQL query
//                    in both verifyAccount() and addAccount()
// Validate Password
Validator pwValidator = new Validator ();
pwValidator.add (new LengthValidator (1, 25) );
if (! pwValidator.isValid (pw)) return false;
// Validate Login Id
Validator loginValidator = new Validator ();
loginValidator.add (new LengthValidator (1, 25) );
loginValidator.add ( new PatternValidator ("^[0-9]+$") );
loginValidator.add (new BoundsValidator (0, Integer.MAX_VALUE) );
if (! loginValidator.isValid (loginId)) return false;
   :
// Code omitted ...
```

A Vulnerability in Depth



A Vulnerability in Depth

- package ait.sanitization;
  - iSanitizer.java

```
public interface iSanitizer {
   public String encode (String sPlain);
   public String decode (String sEncoded);
 }
```

A Vulnerability in Depth

- package ait.sanitization;
  - Sanitizer.java

```
import java.util.*;
public class Sanitizer  extends LinkedList<iSanitizer>
                              implements iSanitizer {
   public String encode (String sPlain) {
      String sEncoded = sPlain;
      Iterator i = iterator();
      while (i.hasNext()) {
        iSanitizer sanitizer = (iSanitizer) i.next();
        sEncoded = sanitizer.encode (sEncoded);
      }
      return sEncoded;
   }
 public String decode (String sEncoded) {
    String sPlain = sEncoded;
    Iterator i = iterator();
    while (i.hasNext()) {
      iSanitizer sanitizer = (iSanitizer) i.next();
      sPlain = sanitizer.decode (sPlain);
    }
    return sPlain;
   }
}
```

Decode should really be performed in reverse order

A Vulnerability in Depth

- package ait.sanitization;
  - SQL_Sanitizer.java

```
public class SQL_Sanitizer extends Sanitizer {
   public String encode (String sPlain) {
     return sPlain.replaceAll ("'", "''");
   }

   public String decode (String sEncoded) {
     return sEncoded.replaceAll ("''", "'");
   }
```

  - In SecureDBAccessor...

```
   // Code omitted ...
   // Sanitization before using input…
   SQL_Sanitizer ss = new SQL_Sanitizer ();
   loginId = ss.encode (loginId);
   pw = ss.encode (pw);
   // Code omitted ...
```

Thank You !

Affinity IT Security Services

Affinity IT Training

IT Security Assessments

Application Security Testing

Secure Development Consulting & Training

www.affinity-it-security.com

(800) 840-2335

For a copy of the slides/examples:

joe@affinity-it.com

Q & A

?