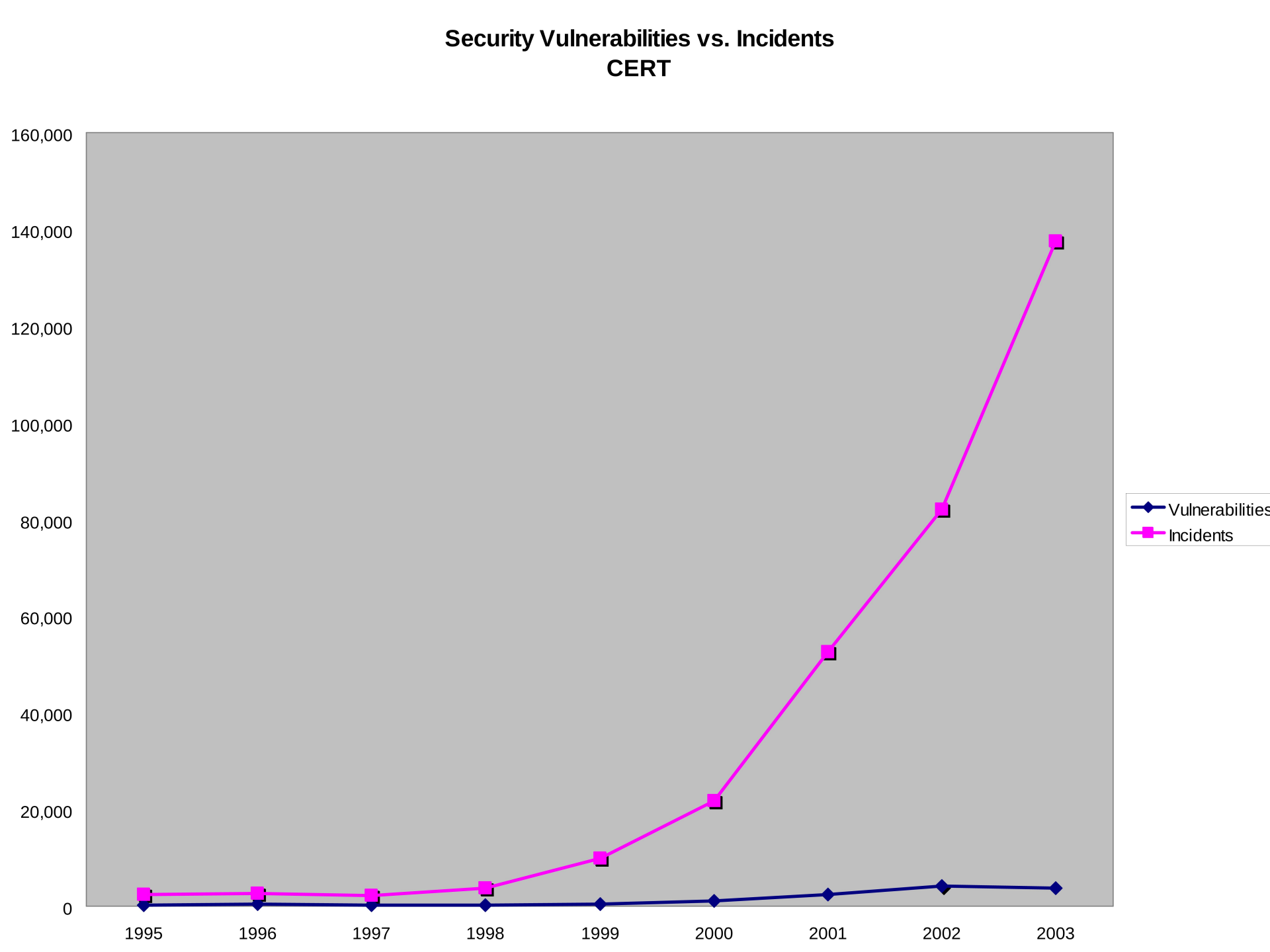
Secure Web Coding w/Java

Martin Nystrom, CISSP
Security Architect
Cisco Systems, Inc.
mnystrom@cisco.com


Security Vulnerabilities vs. Incidents
CERT
160,000
140,000
120,000
100,000
80,000
60,000
40,000
20,000
0
1995
1996
1997
1998
1999
2000
2001
2002
2003
Vulnerabilities
Incidents

Who am I?

- Security Architect in Cisco's InfoSec
  - Responsible for consulting with application teams to secure their architecture
  - Monitor for infrastructure vulnerabilities
  - Infrastructure security architect
- 12 years developing application architectures
- Java programmer
- Master of Engineering – NC State University (2003)
- Bachelor's - Iowa State University – (1990)

Outline

- Introduction to web hacking
- Web architecture
- Principles of secure programming
- Top 10 web vulnerabilities from OWASP

Secure development stats

- Research conducted by MIT and @stake
- Fixing defects during testing is 7 times cheaper than during development
- ROI of fixing bugs

  Testing: 12% ROI

  Implementation: 15% ROI

  Design: 21% ROI

95% of web apps have vulnerabilities

- Cross-site scripting (80 per cent)
- SQL injection (62 per cent)
- Parameter tampering (60 per cent)
- Cookie poisoning (37 per cent)
- Database server (33 per cent)
- Web server (23 per cent)
- Buffer overflow (19 per cent)

Why worry?

- Net worm using Google to spread

  “…uses a flaw in the widely used community forum software known as the PHP Bulletin Board (phpBB) to spread…”

- California reports massive data breach

  “…The compromised system had the names, addresses, phone numbers, social security numbers, and dates of birth of everyone who provided or received care .”

- Google bug exposes e-mail to hackers

  “…By altering the "From" address field of an e-mail sent to the service, hackers could potentially find out a user's personal information, including passwords. ..."

- truckstop.com web application stolen by competitor

  “…Getloaded's officers also hacked into the code Creative used to operate its website.”

Web server attack

- Discover
  - Examine the environment
    - Identify open ports
    - Discover types/versions of apps running
      - Banner grabbing
      - Extensions (.jhtml, .jsp, etc.) and directory structures
  - Generate and examine errors
    - Submit ridiculous input to provoke errors (fuzzing)
    - Database errors, stack traces very helpful
  - Find info left behind (source code, comments, hidden fields)
- Target
  - Login mechanism
  - Input fields
  - Session mgmt
  - Infrastructure

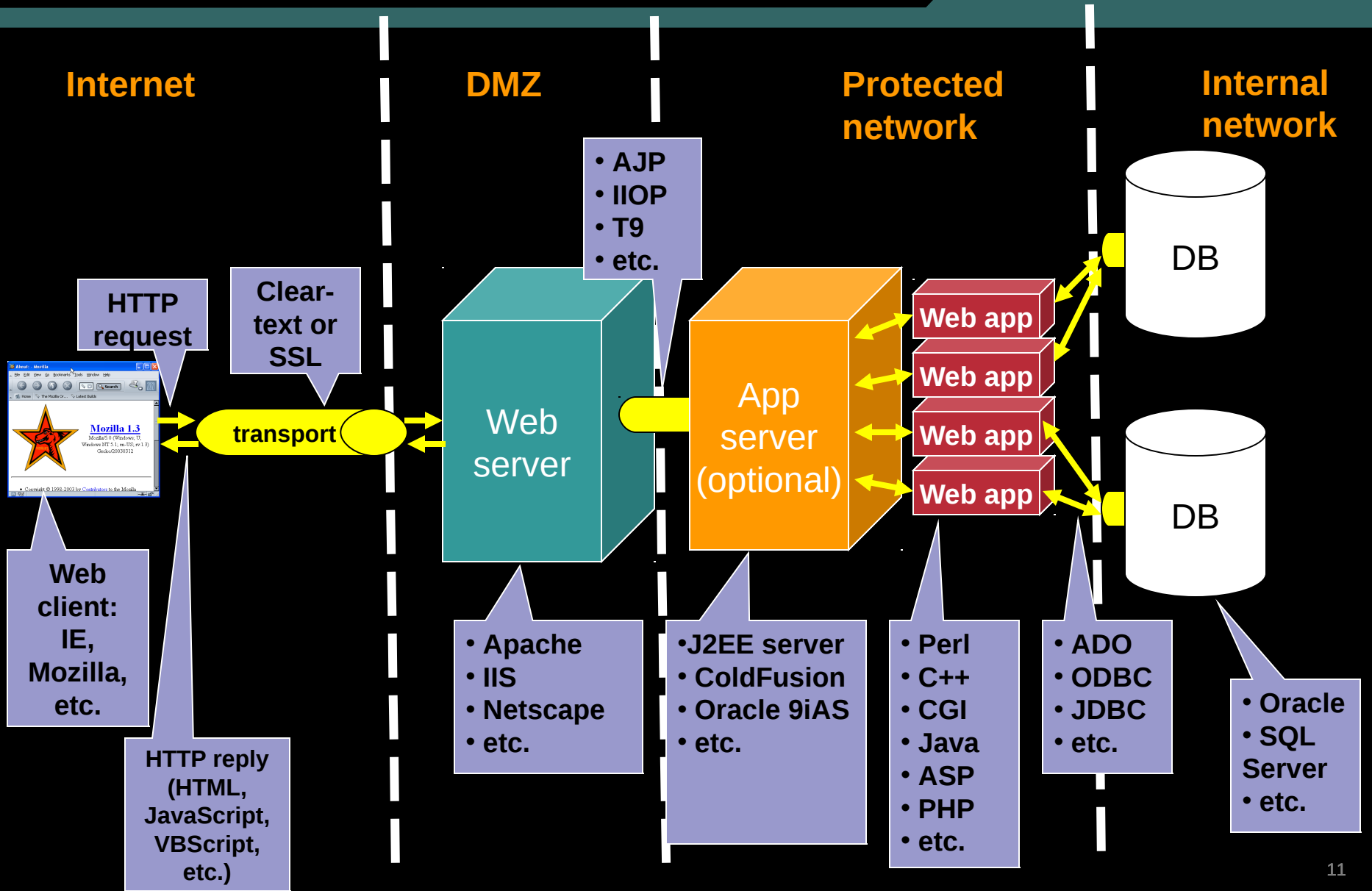
A word of warning

- These tools and techniques can be **dangerous**
- The difference between a hacker and a cracker is… **permission**
- Admins will see strange activity in **logs**, and come looking for you
- Authorities are **prosecuting** even the “good guys” for using these tools

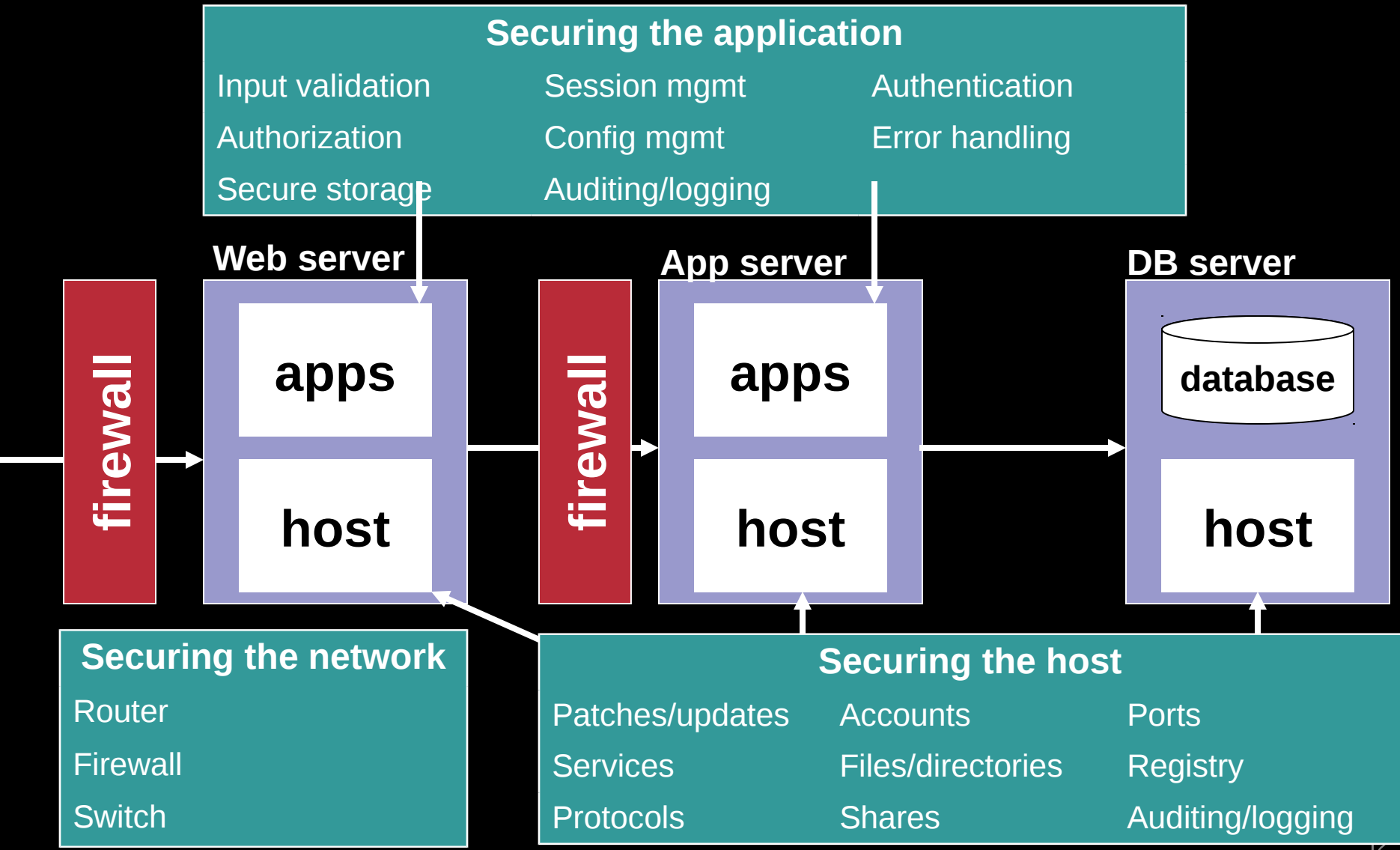
Security principles of web architecture

- Practice defense-in-depth
- Separate services
  - Web server, app server, db server on separate hosts
- Limit privileges of application user
  - File system (chroot or limit privs to read-only)
  - Database system (limit privileges on tables, schemas, etc.)
  - Privileges of running user (xxtomcat, apache, kobayashi, etc.)
- Hide secrets
  - Database account passwords
  - Encryption keys
- Use standard, vetted components, libraries
  - Keep them patched
- Log, and watch logs for unusual activity
- Load-test and tune accordingly

Example web environment



Web Application Security



OWASP Top 10 Web Application Security Vulnerabilities

http://www.owasp.org

1. Unvalidated input
2. Broken access control
3. Broken account/session management
4. Cross-site scripting (XSS) flaws
5. Buffer overflows
6. Injection flaws
7. Improper error handling
8. Insecure storage
9. Denial-of-service
10. Insecure configuration management

Principles for secure coding

- Don’t trust input from user
- Watch for logic holes
- Leverage common, vetted resources
- Only give information needed
- Leverage vetted infrastructure & components
- Build/test to withstand load
  - Expected load
  - Potential DOS attack

#1: Unvalidated Input

- Attacker can easily change any part of the HTTP request before submitting
  - URL
  - Cookies
  - Form fields
  - Hidden fields
  - Headers
- Input must be validated on the server (not just the client).
  - CoolCarts: http://www.extremelasers.com
- Countermeasures
  - Code reviews (check variable against list of allowed values, not vice-versa)
  - Don’t accept unnecessary input from user
  - Store in session or trusted back-end store
  - Sanitize input with regex

#1: Unvalidated input (example)

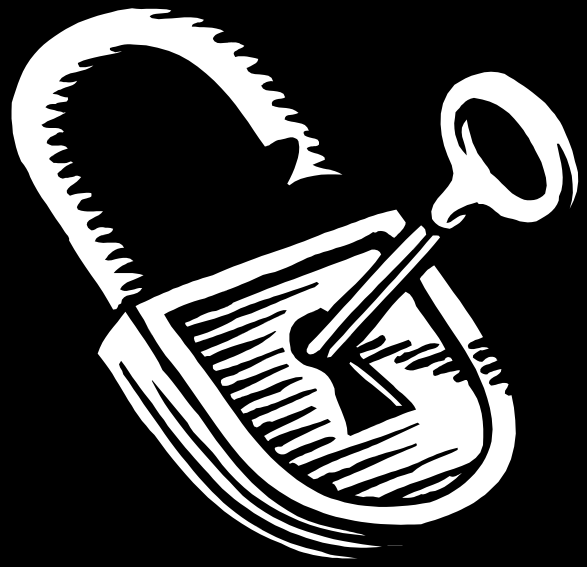
```
public void doPost(HttpServletRequest req,…) {
  String customerId =
    req.getParameter(“customerId”);
  String sku = req.getParameter(“sku”);
  String stringPrice = req.getParameter(“price”);
  Integer price = Integer.valueOf(stringPrice);
  // Store in the database
  orderManager.submitOrder(sku,customerId,price);
} // end doPost
```

#1: Unvalidated input (corrected)

```
public void doPost(HttpServletRequest req,…) {
  // Get customer data
  String customerId =
    req.getParameter(“customerId”);
  String sku = req.getParameter(“sku”);
  // Get price from database
  Integer price = skuManager.getPrice(sku);
  // Store in the database
  orderManager.submitOrder(sku,customerId,price);
} // end doPost
```

#2: Broken access control

- Usually inconsistently defined/applied
- Examples
  - Path traversal
    - Forced browsing past access control checks
    - File permissions – may allow access to config/password files
  - Logic flaws
  - Client-side caching
- Countermeasures
  - Use non-programmatic controls
    - Access control via central container
  - Code reviews



#2: Broken access control (example)

```
protected void doPost(HttpServletRequest req, HttpServletResponse res) {
        try {
           String username = req.getParameter(“USERNAME”);
           String password = req.getParameter(“PASSWORD”);
           try {
             Connection connection = DatabaseUtilities.makeConnection();
             PreparedStatement statement = connection.prepareStatement
               ("SELECT * FROM user_system_data WHERE user_name = ? AND password = ?”);
             statement.setString(1,username);
             statement.setString(2,password);
             ResultSet results = statement.executeQuery(query);
             results.first();
             if (results.getString(1).equals(“”)) {
               s.setMessage("Invalid username and password entered.");
               return (makeLogin(s));
             } // end results check
           } catch (Exception e) {}
           // continue and display the page
           if (username != null && username.length() > 0) {
             return (makeUser(s, username, "PARAMETERS"));
           } // end username test
        } catch (Exception e) {
           s.setMessage("Error generating " + this.getClass().getName());
        } // end try/catch
        return (makeLogin(s));
} // end doPost
```

#2: Broken access control (solution)
How to set up basic authentication on CCX

```
<security-constraint>
  <web-resource-collection>
     <web-resource-name>Admin</web-resource-name>
     <url-pattern>/jsp/admin/*</url-pattern>
  </web-resource-collection>
  <auth-constraint>
    <role-name>(accessLevel=4)</role-name>
  </auth-constraint>
</security-constraint>
<login-config>
  <auth-method>BASIC</auth-method>
  <realm-name>CCO</realm-name>
</login-config>
```

#2: Broken access control (solution)
How to set up form authentication on CCX

web.xml file

```
<!-- LOGIN AUTHENTICATION -->
  <login-config>
    <auth-method>FORM</auth-method>
    <realm-name>CCO</realm-name>
    <form-login-config>
      <form-login-page>login.jsp</form-login-page>
      <form-error-page>error.jsp</form-error-page>
    </form-login-config>
  </login-config>
```

login.jsp

```
<form method="POST"  action= "j_security_check" >
 <input type="text"  name= "j_username" >
 <input type="password"  name= "j_password" >
</form>
```

#3: Broken Account and Session Management

- Weak user authentication
  - Password-only
  - Easily guessable usernames (admin, etc.)
  - Poorly implemented single sign-on (SSO)
- Weak resource authentication
  - How are database passwords stored?
  - Review trust relationships between hosts
    - IP address can be spoofed, etc.
- Countermeasures
  - Use vetted single sign-on and session mgmt solution
    - Netegrity SiteMinder
    - RSA ClearTrust
  - Strong passwords
  - Remove default user names
  - Protect sensitive files

#3: Broken account/session management (client example - SSO)

```
public void doGet(HttpServletRequest req,…) {
  // Get user name
  String userId = req.getRemoteUser();
  Cookie ssoCookie = new Cookie(“userid”,userId);
  ssoCookie.setPath(“/”);
  ssoCookie.setDomain(“cisco.com”);
  response.addCookie(ssoCookie);
  …
}
```

#3: Broken account/session management (server example - SSO)

```
public void doGet(HttpServletRequest req,…) {
  // Get user name
  Cookie[] cookies = req.Cookies();
  for (i=0; i < cookies.length; i++) {
    Cookie cookie = cookies[i];
    if (cookie.getName().equals(“ssoCookie”)) {
       String userId = cookie.getValue();
       HttpSession session = req.getSession();
       session.setAttribute(“userId”,userId);
    } // end if
  } // end for
} // end doGet
```

#3: Broken account/session management (client solution - SSO)

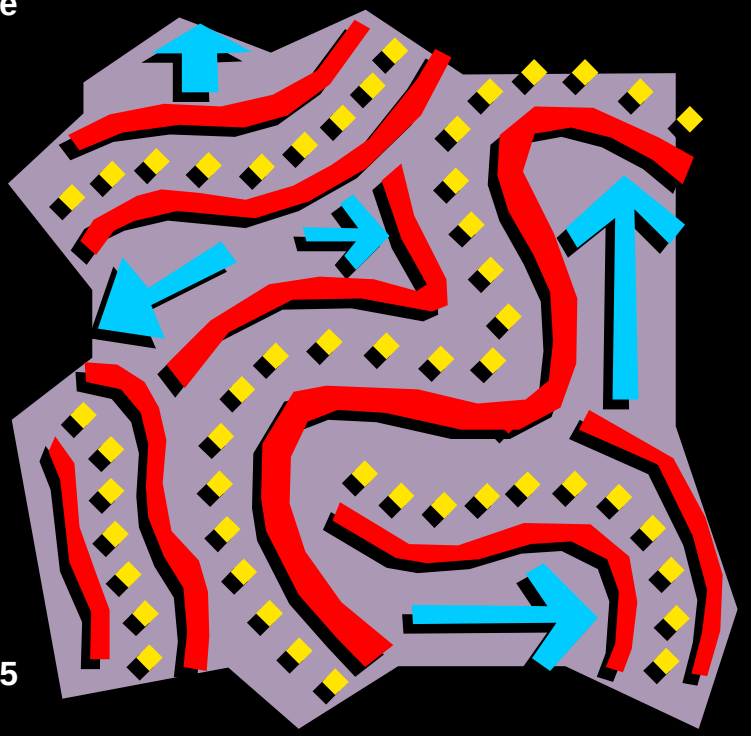
```
public void doGet(HttpServletRequest req,…) {
  // Get user name
  String userId = req.getRemoteUser();
  encryptedUserId = Encrypter.encrypt(userId);
  Cookie ssoCookie =
    new Cookie(“userid”,encrypteduserId);
  ssoCookie.setPath(“/”);
  ssoCookie.setDomain(“cisco.com”);
  response.addCookie(ssoCookie);
  …
}
```

#3: Broken account/session management (server solution - SSO)

```
public void doGet(HttpServletRequest req,…) {
  // Get user name
  Cookie[] cookies = req.Cookies();
  for (i=0; i < cookies.length; i++) {
    Cookie cookie = cookies[i];
    if (cookie.getName().equals(“ssoCookie”)) {
        String encryptedUserId = cookie.getValue();
        String userId = Encrypter.decrypt(encryptedUserId);
        if (isValid(userId)) {
            HttpSession session = req.getSession();
            session.setAttribute(“userId”,userId);
        }
    } // end if
  } // end for
} // end doGet
```

#4: Cross-Site Scripting (XSS)

- **Attacker…**
  - **Inject code into web page that is then displayed to user in the browser**
  - **Uses trusted application/company to reflect malicious code to end-user**
  - **Can “hide” the malicious code w/unicode**
- **Vulnerable anywhere user-supplied data is redisplayed w/out input validation or output encoding**
- **2 types of attacks: stored & reflected**
- **Can steal cookies, especially vulnerable on apps with form-based authentication**
- **Countermeasures**
  - **Input validation**
    - **White-listing: a-z, A-Z, 0-9, etc.)**
    - **Black-listing: “< > ( ) # &”**
    - **Don’t forget these: “&lt &gt &#40 &#41 &#35 &#38”**
  - **Output encoding (htmlEncode output)**
  - **Truncate input fields to reasonable length**



#4: Cross-site scripting (flaw)

```
protected void doPost(HttpServletRequest req, HttpServletResponse res) {
  String title = req.getParameter("TITLE");
  String message = req.getParameter("MESSAGE");
  try {
    connection = DatabaseUtilities.makeConnection(s);
    PreparedStatement statement =
      connection.prepareStatement
            ("INSERT INTO messages VALUES(?,?)");
    statement.setString(1,title);
    statement.setString(2,message);
    statement.executeUpdate();
  } catch (Exception e) {
    …
  } // end catch
} // end doPost
```

#4: Cross-site scripting (solution)

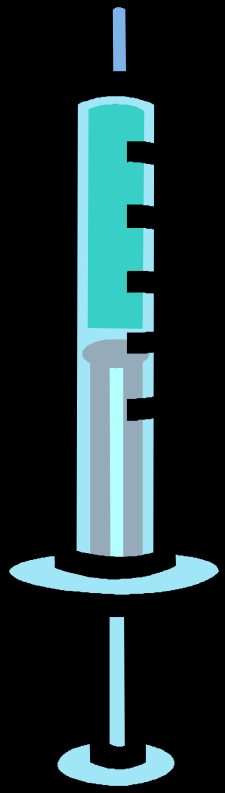
```
private static String stripEvilChars(String evilInput) {
  Pattern evilChars = Pattern.compile("[^a-zA-Z0-9]");
  return evilChars.matcher(evilInput).replaceAll("");
}

protected void doPost(HttpServletRequest req, HttpServletResponse res) {
  String title = stripEvilChars(req.getParameter("TITLE"));
  String message = stripEvilChars(req.getParameter("MESSAGE"));
  try {
    connection = DatabaseUtilities.makeConnection(s);
    PreparedStatement statement =
      connection.prepareStatement
              ("INSERT INTO messages VALUES(?,?)");
    statement.setString(1,title);
    statement.setString(2,message);
    statement.executeUpdate();
  } catch (Exception e) {
    …
  } // end catch
} // end doPost
```

#5 Buffer overflow errors

- **Not generally an issue with Java apps**
- **Avoid use of native methods**
  - **Especially from untrusted sources**

#6: Injection flaws

- Allows attacker to relay malicious code in form variables or URL
  - System commands
  - SQL
- Typical dangers
  - `Runtime.exec()` to external programs (like sendmail)
  - Dynamically concatenated SQL statements
- Examples
  - Path traversal: "`../`"
  - Add more commands: "`; rm –r *`"
  - SQL injection: "`' OR 1=1`"
- Countermeasures
  - Use PreparedStatements in SQL
  - Avoid Runtime.exec() calls (use libraries instead)
  - Run with limited privileges
  - Filter/validate input



#6: SQL injection (flaw)

```
protected void doPost(HttpServletRequest req, HttpServletResponse res) {
  String query =
    "SELECT userid, name FROM user_data WHERE accountnum = '"
    + req.getParameter("ACCT_NUM")
    + "'";
  PrintWriter out = res.getWriter();
  // HTML stuff to out.println…
  try {
    connection = DatabaseUtilities.makeConnection(s);
    Statement statement = connection.createStatement();
    ResultSet results = statement.executeQuery(query);
    while (results.next ()) {
        out.println("<TR><TD>" + rset.getString(1) + "</TD>");
  out.println("<TD>" + rset.getString(2) + "</TD>");
    } // end while
  } catch (Exception e) {
  // exception handling…
  } // end catch
} // end doPost
```

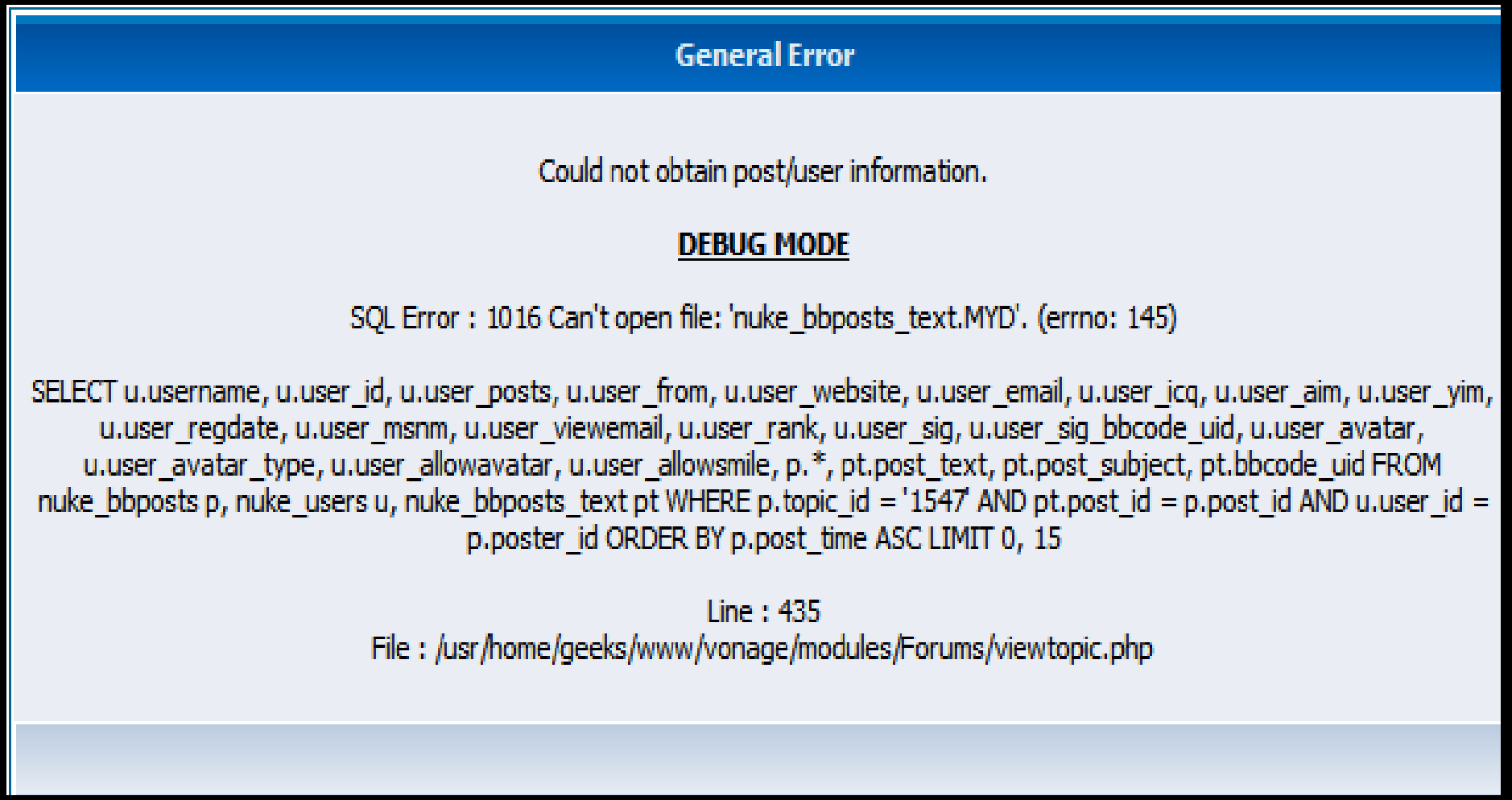
#6: SQL injection (fix)

```
protected void doPost(HttpServletRequest req, HttpServletResponse res) {
  PrintWriter out = res.getWriter();
  // HTML stuff to out.println…
  try {
    connection = DatabaseUtilities.makeConnection(s);
    PreparedStatement statement = connection.prepareStatement
  ("SELECT userid, name FROM user_data WHERE accountnum = ?“);
    statement.setString(1,req.getParameter(“ACCT_NUM”);
    ResultSet results = statement.executeQuery(query);
    while (results.next ()) {
       out.println("<TR><TD>“ + rset.getString(1) + “</TD>”);
  out.println("<TD>“ + rset.getString(2) + “</TD>”);
    } // end while
  } catch (Exception e) {
  // exception handling…
  } // end catch
} // end doPost
```

#7: Improper error handling

- Examples: stack traces, DB dumps
- Helps attacker know how to target the app
- Often left behind during programmer debugging
- Inconsistencies can be revealing
  - "File not found" vs. "Access denied"
- Gives insight into source code
  - Logic flaws
  - Default accounts, etc.
- Good messages give enough info to user w/o giving too much info to attacker
- Countermeasures
  - Code review
  - Modify default error pages (404, 401, etc.)
  - Log details to log files, not returned in HTTP request

Error messages example



General Error

Could not obtain post/user information.

DEBUG MODE

SQL Error : 1016 Can't open file: 'nuke_bbposts_text.MYD'. (errno: 145)

SELECT u.username, u.user_id, u.user_posts, u.user_from, u.user_website, u.user_email, u.user_icq, u.user_aim, u.user_yim, u.user_regdate, u.user_msnm, u.user_viewemail, u.user_rank, u.user_sig, u.user_sig_bbcode_uid, u.user_avatar, u.user_avatar_type, u.user_allowavatar, u.user_allowsmile, p.*, pt.post_text, pt.post_subject, pt.bbcode_uid FROM nuke_bbposts p, nuke_users u, nuke_bbposts_text pt WHERE p.topic_id = '1547' AND pt.post_id = p.post_id AND u.user_id = p.poster_id ORDER BY p.post_time ASC LIMIT 0, 15

Line : 435
File : /usr/home/geeks/www/vonage/modules/Forums/viewtopic.php

#7: Improper error handling (flaw)

```
protected void doPost(HttpServletRequest req, HttpServletResponse res) {
  String query =
    "SELECT userid, name FROM user_data WHERE accountnum = '"
    + req.getParameter("ACCT_NUM") + "'";
  PrintWriter out = res.getWriter();
  // HTML stuff to out.println…
  try {
    connection = DatabaseUtilities.makeConnection(s);
    Statement statement = connection.createStatement();
    ResultSet results = statement.executeQuery(query);
    while (results.next ()) {
        out.println("<TR><TD>" + rset.getString(1) + "</TD>");
  out.println("<TD>" + rset.getString(2) + "</TD>");
    } // end while
  } catch (Exception e) {
  e.printStackTrace(out);
  } // end catch
} // end doPost
```

#7: Improper error handling (solution)

```
protected void doPost(HttpServletRequest req, HttpServletResponse res) {
  String query =
    "SELECT userid, name FROM user_data WHERE accountnum = '"
    + req.getParameter("ACCT_NUM") + "'";
  PrintWriter out = res.getWriter();
  // HTML stuff to out.println…
  try {
    connection = DatabaseUtilities.makeConnection(s);
    Statement statement = connection.createStatement();
    ResultSet results = statement.executeQuery(query);
    while (results.next ()) {
        out.println("<TR><TD>" + rset.getString(1) + "</TD>");
        out.println("<TD>" + rset.getString(2) + "</TD>");
    } // end while
  } catch (Exception e) {
        Logger logger = Logger.getLogger();
        logger.log(Level.SEVERE,"Error retrieving account number",e);
        out.println("Sorry, but we are unable to retrieve this account");
  } // end catch
} // end doPost
```

#8: Insecure storage

- Sensitive data such as credit cards, passwords, etc. must be protected
- Examples of bad crypto
  - Poor choice of algorithm
  - Poor randomness in sessions/tokens
- Storage locations must be protected
  - Database
  - Files
  - Memory
- Countermeasures
  - Store only what you must
  - Store a hash instead of the full value if you can (SHA-1, for example)
  - Use only vetted, public cryptography

#8: Insecure storage – bad example

```
public String encrypt(String plainText) {
  plainText = plainText.replace(“a”,”z”);
  plainText = plainText.replace(“b”,”y”);
  …
  return Base64Encoder.encode(plainText);
}
```

#8: Insecure storage – fixed example

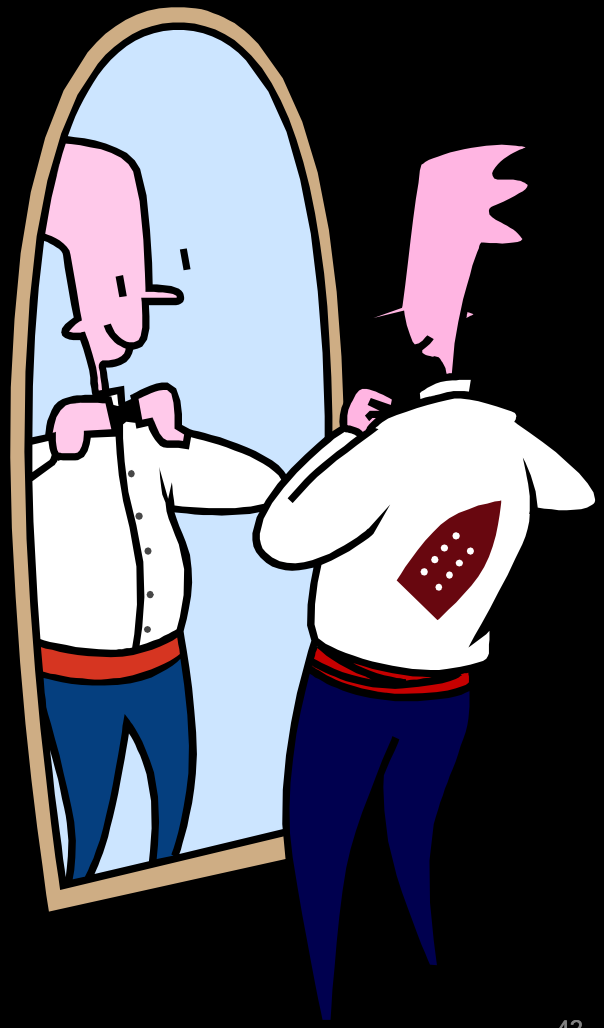
```
public String encrypt(String plainText) {
  // Read encryptKey as a byte array from a file
  DESKeySpec keySpec = new DESKeySpec(encryptKey);
  SecretKeyFactory factory =
      new SecretKeyFactory.getInstance(“DES”);
  SecretKey key = factory.generateSecret(keySpec);
  Cipher cipher = Cipher.getInstance(“DES”);
  cipher.init(Cipher.ENCRYPT_MODE,key);
  byte[] utf8text = plainText.getBytes(“UTF8”);
  byte[] enryptedText = ecipher.doFinal(utf8text);
  return Base64Encoder.encode(encryptedText);
}
```

#9: Denial-of-service (DoS)

- Examples that may provoke DoS
  - Heavy object allocation/reclamation
  - Overuse of logging
  - Unhandled exceptions
  - Unresolved dependencies on other systems
  - Web services
  - Databases
- May impact other applications, hosts, databases, or network itself
- Countermeasures
  - Load testing
  - Code review

#10: Insecure configuration management

- Tension between “work out of the box” and “use only what you need”
- Developers ≠ web masters
- Examples
  - Unpatched security flaws (BID example)
  - Misconfigurations that allow directory traversal
  - Administrative services accessible
  - Default accounts/passwords
- Countermeasures
  - Create and use hardening guides
  - Turn off all unused services
  - Set up and audit roles, permissions, and accounts
  - Set up logging and alerts



Principles for secure coding

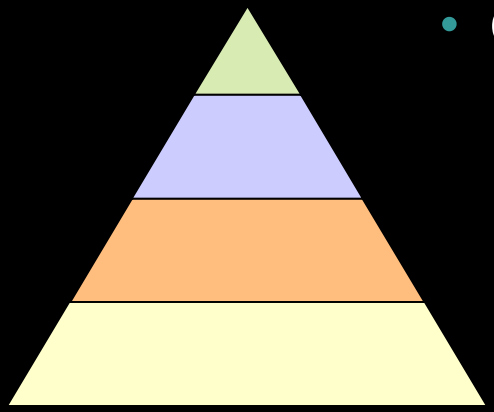
- Don’t trust input from user
- Watch for logic holes
- Leverage common, vetted resources
- Only give information needed
- Leverage vetted infrastructure & components
- Build/test to withstand load
  - Expected load
  - Potential DOS attack

Tools used in this preso

- WebGoat –vulnerable web applications for demonstration
- VMWare – runs Linux & Windows 2000 virtual machines on demo laptop.
- nmap –host/port scanning to find vulnerable hosts
- Mozilla Firefox – browser that supports plug-ins for proxied HTTP, source browsing
  - SwitchProxy plug-in lets you quickly switch your proxies
  - WebDeveloper plug-in lets you easily clear HTTP auth
- WebScarab – HTTP proxy

Backup slides & old slides

#9: Remote Administration Flaws



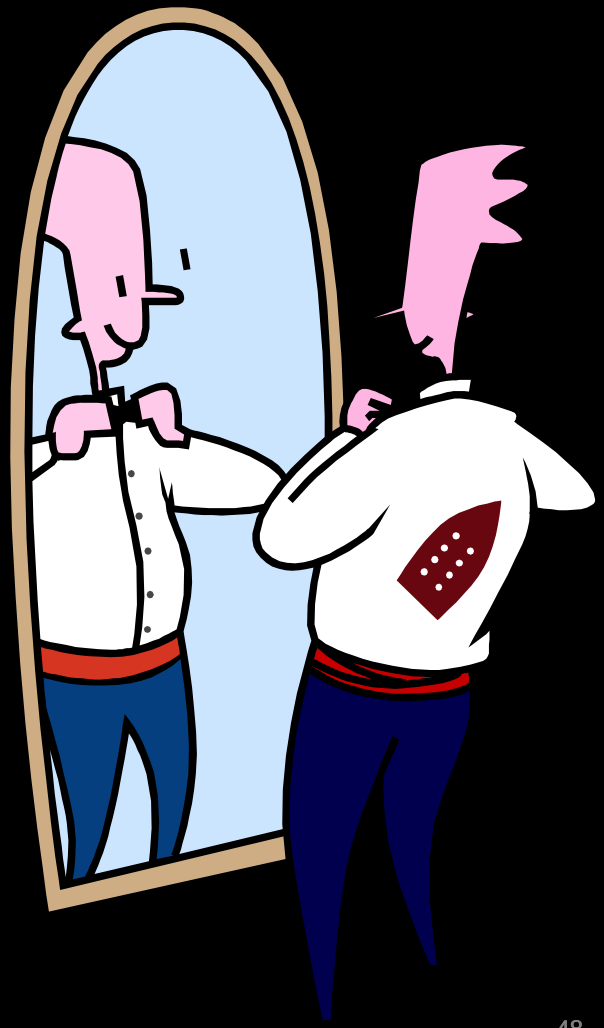
- **Problems**
  - **Weak authentication (username=“admin”)**
  - **Weak encryption**
- **Countermeasures**
  - **Don’t place admin interface on same server**
  - **Use strong authentication: certificates, tokens, strong passwords, etc.**
  - **Encrypt entire session (VPN or SSL)**
  - **Control who has accounts**
  - **IP restrictions**

#7: Fail open authentication – code fix

```
protected void doPost(HttpServletRequest req, HttpServletResponse res) {
      try {
      String username = req.getParameter(“USERNAME”);
      String password = req.getParameter(“PASSWORD”);
      try {
        Connection connection = DatabaseUtilities.makeConnection();
        PreparedStatement statement = connection.prepareStatement
          ("SELECT * FROM user_system_data WHERE user_name = ? AND password = ?”);
        statement.setString(1,username);
        statement.setString(2,password);
        ResultSet results = statement.executeQuery(query);
        if (results == null || !results.first()) {
          s.setMessage("Invalid username and password entered.");
          return (makeLogin(s));
        } // end results check
      } catch (Exception e) {}
      // continue and display the page
      if (username != null && username.length() > 0) {
           return (makeUser(s, username, "PARAMETERS"));
      } // end username test
      } catch (Exception e) {
      s.setMessage("Error generating " + this.getClass().getName());
      } // end try/catch
      return (makeLogin(s));
} // end doPost
```

#10: Insecure configuration management

- Tension between “work out of the box” and “use only what you need”
- Developers ≠ web masters
- Examples
  - Unpatched security flaws (BID example)
  - Misconfigurations that allow directory traversal
  - Administrative services accessible
  - Default accounts/passwords
- Countermeasures
  - Create and use hardening guides
  - Turn off all unused services
  - Set up and audit roles, permissions, and accounts
  - Set up logging and alerts



#3 bad example of session id generation

```
// Tomcat version 1 (JServ)
//
public class SessionIdGenerator {
    private static int counter = 1010;
    public static synchronized String generateId() {

        Integer i = new Integer(counter++);
        StringBuffer buf = new StringBuffer();
        String dString = Double.toString(Math.abs(Math.random()));

        buf.append("To");
        buf.append(i);
        buf.append("mC");
        buf.append(dString.substring(2, dString.length()));
        buf.append("At");

        return buf.toString();
    }
}
```

#3 fixed example session id generation

```
public class SessionIdGenerator {

    static private int session_count = 0;
    static private long lastTimeVal = 0;
    static private java.util.Random randomSource = new java.security.SecureRandom();

    // MAX_RADIX is 36
    public final static long maxRandomLen = 2176782336L; // 36 ** 6

    public final static long maxSessionLifespanTics = 46656; // 36 ** 3

public final static long ticDifference = 2000;

static synchronized public String getIdentifier (String jsIdent)
    {
        StringBuffer sessionId = new StringBuffer();

        // random value ..
        long n = randomSource.nextLong();
        if (n < 0) n = -n;
        n %= maxRandomLen;
n += maxRandomLen;
        sessionId.append (Long.toString(n, Character.MAX_RADIX)
                .substring(1));

        long timeVal = (System.currentTimeMillis() / ticDifference);
        // cut..
        timeVal %= maxSessionLifespanTics;
        // padding, see above
        timeVal += maxSessionLifespanTics;

        sessionId.append (Long.toString (timeVal, Character.MAX_RADIX)
                .substring(1));

if (lastTimeVal != timeVal) {
          lastTimeVal = timeVal;
          session_count = 0;
        }
        sessionId.append (Long.toString (++session_count,
                  Character.MAX_RADIX));

        if (jsIdent != null && jsIdent.length() > 0) {
          return sessionId.toString()+"."+jsIdent;
        }
        return sessionId.toString();
    }

    public static synchronized String generateId() {
        return getIdentifier(null);
    }
}
```

#4: Bad example – output vuln to XSS

```
protected void doGet(HttpServletRequest req, HttpServletResponse res) {
  int messageNum = Integer(res.getParameter(NUMBER)).intValue();
  String title, message;
  try {
    PreparedStatement statement = connection.prepareStatement
      ("SELECT title,message FROM messages WHERE num = ?“);
    statement.setInt(1,messageNum);
    ResultSet results = statement.executeQuery(query);
    title = results.getString(1);
    message = results.getString(2);
  } catch(Exception e) {
    // exception handling
  }
  PrintWriter out = response.printWriter();
  res.setContentType(“text/html”);
  PrintWriter out = res.getWriter();
  // HTML page formatting
  out.println(title + “ - “ + message);
}
```

#4: Good example – output vuln to XSS

```
protected void doGet(HttpServletRequest req, HttpServletResponse res) {
  int messageNum = Integer(res.getParameter(NUMBER)).intValue();
  String title, message;
  try {
     PreparedStatement statement = connection.prepareStatement
       ("SELECT title,message FROM messages WHERE num = ?“);
     statement.setInt(1,messageNum);
     ResultSet results = statement.executeQuery(query);
     title = results.getString(1);
     message = results.getString(2);
   } catch(Exception e) {
     // exception handling
   }
  PrintWriter out = response.printWriter();
   res.setContentType(“text/html”);
   PrintWriter out = res.getWriter();
  // HTML page formatting
  out.println(htmlEncode(title + “ - “ + message));
}
```