



中山大學  
SUN YAT-SEN UNIVERSITY

# Security Testing



## SE-307 Software Testing Techniques

<http://my.ss.sysu.edu.cn/wiki/display/SE307/Home>

Instructor: Dr. Wang Xinming, School of Software, Sun Yat-Sen University

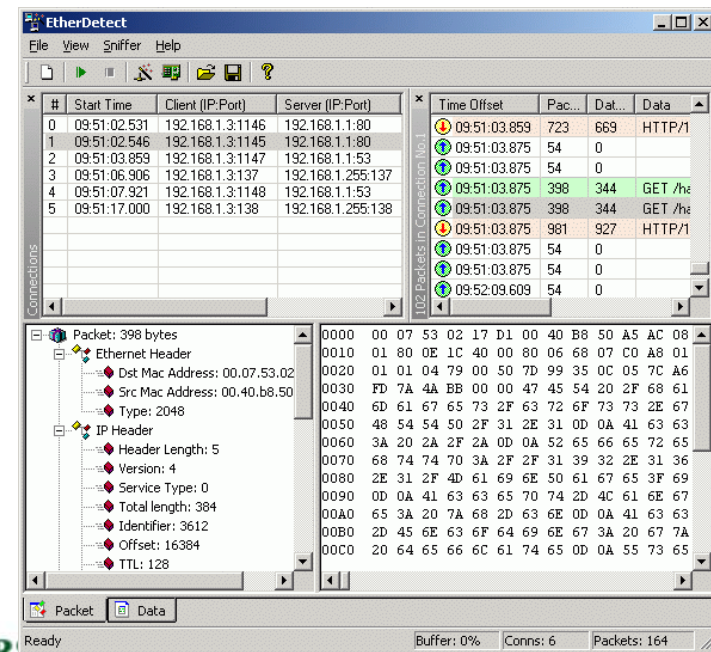
# Denial-of-Service (DoS)

---

- **Denial of Service (DoS) attack:**  
Attacker causes web server to be unavailable.
- How attack is performed:
  - Attacker frequently requests many pages from your web site.
    - **distributed DoS (DDoS):** DoS using lots of computers
  - Your server cannot handle this many requests at a time, so it turns into a smoldering pile of goo (or just becomes very slow).
- Problems that this attack can cause:
  - Users cannot get to your site.
  - Online store's server crashes -> store loses potential revenue.
  - Server may crash and lose or corrupt important data.
  - All the bandwidth used by the DoSers may cost you \$\$\$.

# Packet sniffing

- **packet sniffing:** Listening to traffic sent on a network.
  - Many internet protocols (http, aim, email) are unsecure.
  - If an attacker is on the same local network (LAN) as you, he can:
    - read your email/IMs as you send them
    - see what web sites you are viewing
    - grab your password as it's being sent to the server
- solutions:
  - Use secure protocols (ssh, https)
  - Encryption
  - Don't let creeps on your LAN/wifi



# Password cracking

---

- **password cracking:**  
Guessing the passwords of privileged users of your system.
- How attack is performed:
  - **brute force attack:** Attacker uses software that sequentially tries every possible password.
  - **dictionary attack:** Attacker uses software that sequentially tries passwords based on words in a dictionary.
    - every word in the dictionary
    - combinations of words, numbers, etc.
- What you can do about it:
  - Force users to have secure passwords.
  - Block an IP address from logging in after N failed attempts.

# Phishing/social engineering

- **phishing:** Masqueraded mails or web sites.
  - **social engineering:** Attempts to manipulate users, such as fraudulently acquiring passwords or credit card numbers.
- Problems:
  - If trusted users of your system are tricked into giving out their personal information, attackers can use this to log in as those users and compromise your system.



## Security Center Advisory!

We recently noticed one or more attempts to log in to your PayPal account from a foreign IP address and we have reasons to believe that your account was hijacked by a third party without your authorization. If you recently accessed your account while traveling, the unusual log in attempts may have been initiated by you.

If you are the rightful holder of the account you must **click the link below** and then complete all steps from the following page as we try to verify your identity.

[Click here to verify your account](#)

[http://211.248.156.177/.PayPal/cgi-bin/websecrcmd\\_login.php](http://211.248.156.177/.PayPal/cgi-bin/websecrcmd_login.php)

If you choose to ignore our request, you leave us no choice but to temporarily suspend your account.

## Protect Your Account Info

Make sure you never provide your password to fraudulent persons.

PayPal automatically encrypts your confidential information using the Secure Sockets Layer protocol (SSL) with an encryption key length of 128-bits (the highest level commercially available).

PayPal will never ask you to enter your password in an email.

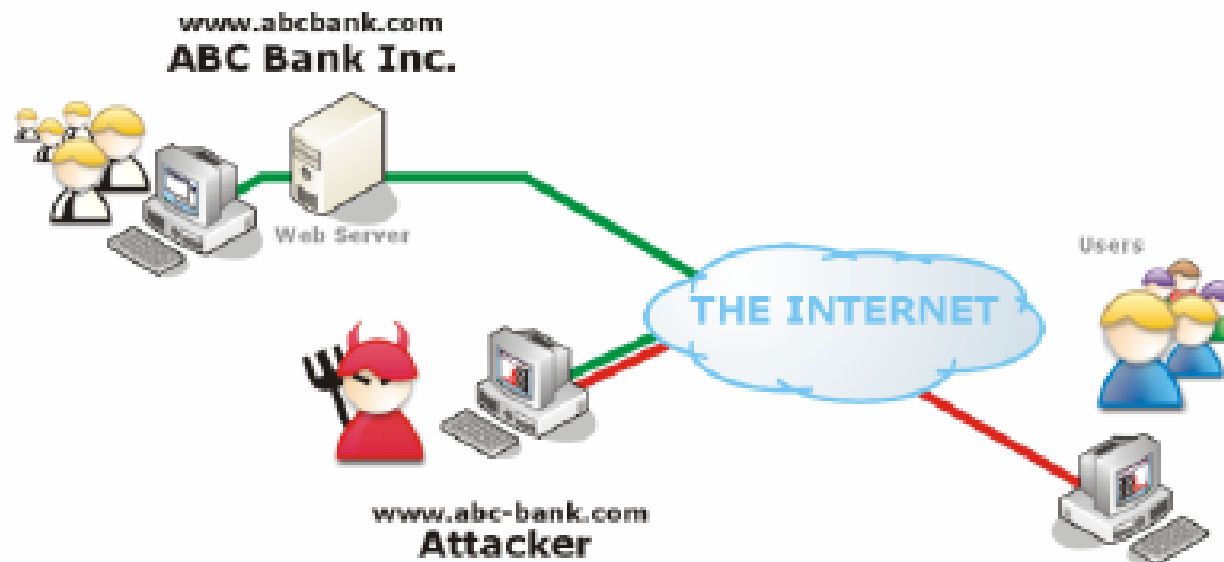
For more information on protecting yourself from fraud, please review our Security Tips at <http://www.paypal.com/securitytips>

## Protect Your Password

# Man-in-the-middle

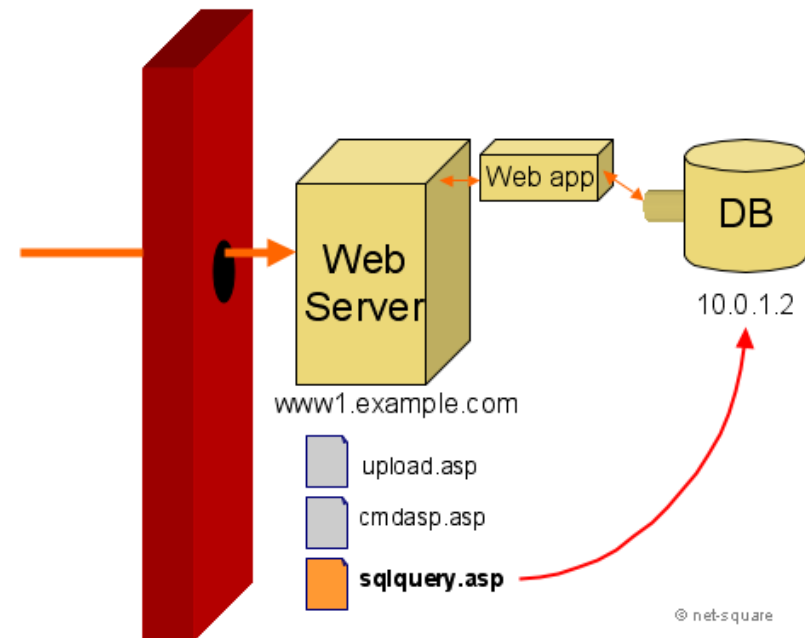
---

- **man-in-the-middle attack:** Attacker sits between two machines and silently intercepts traffic between them.
  - tricks user to go to attacker's site instead of real site
  - intercepts sensitive information and/or modifies data before sending it from one endpoint to the other



# Privilege escalation

- **privilege escalation:** Attacker becomes able to run code on your server as a privileged user.
  - Example: Perhaps normal users aren't able to directly query your database. But an attacker may find a flaw in your security letting him run as an administrator and perform the query.
  - Once you're running as root, You own the server. You can do anything you want!



# Cross-site scripting (XSS)

---

- **cross-site scripting:** Causing one person's script code to be executed when a user browses to another site.
  - Example: Visit google.com, but evil.com's JavaScript runs.
- How attack is performed:
  - Attacker finds unsecure code on target site.
  - Attacker uses hole to inject JavaScript into the page.
  - User visits page, sees malicious script code.





# SQL Injection

- **SQL injection:**

Causing undesired SQL queries to be run on your database.

- Often caused when untrusted input is pasted into a SQL query

PHP: "SELECT \* FROM Users WHERE name=' \$name ' ; " ;

- specify a user name of: **x' OR 'a'='a**

SELECT \* FROM Users WHERE name=' **x' OR**

**'a'='a'** ;



# Panning for gold

- **View Source**, and look for:
  - HTML comments
  - script code
  - other sensitive information in code:  
IP/email addresses, SQL queries, hidden fields,...
- watch HTTP requests/responses
  - look for hidden pages, files, parameters to target
- error messages sent to your browser by app
 

● 200: OK	400: Invalid request
● 403: Forbidden	404: File not found
● 500: Internal server error	



```

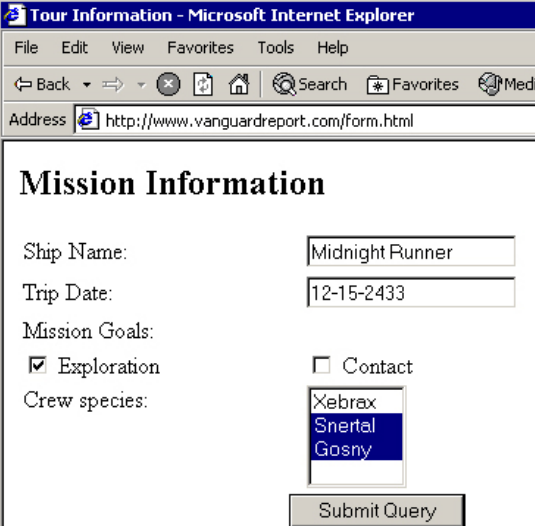
<html xmlns="http://www.w3.org/1996/xhtml" >
  <head><meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <meta name="Generator" content="iWeb-Build" />
    <meta http-equiv="X-UA-Compatible" content="IE=8" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <title></title>
    <link rel="stylesheet" type="text/css" href="files/huan_ying.css" />
    <!--[if lt IE 8]><link rel="stylesheet" type="text/css" href="huan_ying_files/huan_yingIE8.css" /><![endif]>
    <!--[if gte IE 8]><link rel="stylesheet" type="text/css" href="Media/IE8.css" /><![endif]>
    <script type="text/javascript" src="common.js"></script>
    <script type="text/javascript" src="pt"></script>
    <script type="text/javascript">
    <script type="text/javascript">
    </head>
    <body style="background: rgb(255, 255, 255); color: black; font-family: sans-serif; font-size: 12px; text-align: center; padding: 10px; overflow: hidden; background-color: rgb(255, 255, 255); text-align: center;">
      <div style="margin-bottom: 10px; margin-top: 0px; overflow: hidden; padding: 0px; background-color: rgb(255, 255, 255); text-align: center;">
        <div style="margin-left: 0px; margin-right: 0px; height: 0px; width: 100%; border-top: 1px solid black; border-bottom: 1px solid black; border-left: 1px solid black; border-right: 1px solid black;">

```

# Input forms

---

- **Forms** let users pass parameters to the web server.
- Parameters are passed using GET or POST requests.
  - **GET**: parameters are contained in the request URL.  
`http://www.google.com?q=Stephen+Colbert&lang=en`
  - **POST**: parameters are contained in the HTTP packet header.
    - harder for the user to see, but no more secure than GET
- Forms provide a rich attack ground...



The screenshot shows a Microsoft Internet Explorer window titled "Tour Information - Microsoft Internet Explorer". The address bar displays "http://www.vanguardreport.com/form.html". The page content includes a form titled "Mission Information" with the following fields:

- Ship Name:
- Trip Date:
- Mission Goals: ☒ Exploration ☐ Contact
- Crew species:  (dropdown menu showing "Xebrax", "Snertal", and "Gosny")
-

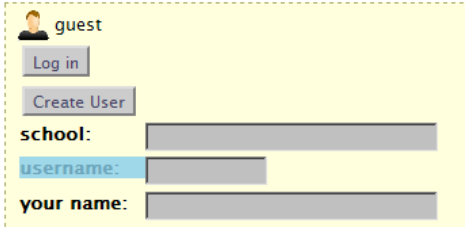
# Form validation

- **validation:** Examining form parameters to make sure they are acceptable before/as they are submitted.
  - nonempty, alphabetical, numeric, length, ...
  - **client-side:** HTML/JS checks values before request is sent.
  - **server-side:** JSP/Ruby/PHP/etc. checks values received.
- Some validation is performed by restricting the user's choices.
  - select boxes
  - input text boxes with `maxlength` attribute
  - key event listeners that erase certain key presses



# User input attacks

- Bypassing client-side input restrictions and validation
  - maxlength limits on an input text field
  - choices not listed in a select box
  - hidden input fields
  - modifying or disabling client-side JavaScript validation code



**Building Java Programs : Practice-It!**

Choose a category and problem:

**BJP Chapter Exercises**

Chapter 1: Introduction to Java Programming (29)

Chapter 2: Primitive Data and Definite Loops (39)

Chapter 3: Parameters and Objects (26)

Chapter 3G Supplement: Graphics (9)

Chapter 4: Conditional Execution (17)

Console HTML CSS Script DOM Net

Edit | span#allschools

```

<div id="createschoolcompleter" style="display: none;"/>
</div>
<div>
  <label class="columnname" for="createusername">username:</label>
  <input id="createusername" type="text" maxlength="10" size="12" name="username"/>
</div>
<div>
  <label class="columnname" for="createname">your name:</label>
  <input id="createname" type="text" maxlength="30" size="30" name="name"/>
</div>

```

# Guessing files/directories

---

- **security through obscurity:** Many reachable files/resources are hidden only by the fact that there is no link to them.
- Try common file/folder/commands to see what happens:
  - `/etc/passwd` , `/etc/shadow` , `cat` , `ls` , `grep`
  - guess file names based on others
    - `page11.php`                      --> `page12.php`
    - `loginfailure.jsp`                --> `loginsuccess.jsp`
    - `accounts/fred.html`              --> `accounts/sue.html`
  - brute force / web spiders
  - port scanners



# Other attacks

---

- Attacking GET parameters
- Attacking hidden input fields
- Attacking cookies
- Cross-site request forgery (CSRF)
- ...



# Methods of security

---

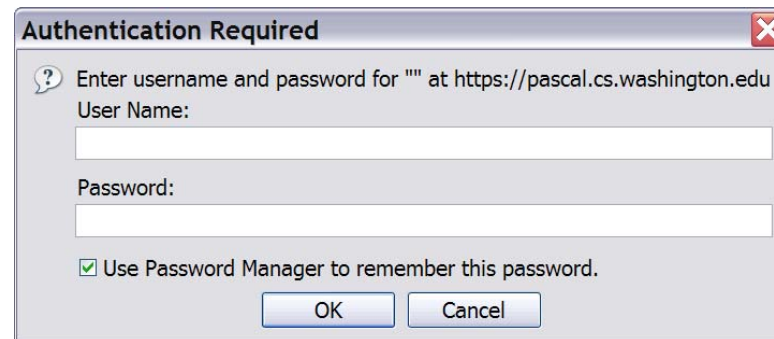
- **Security through obscurity:** Relying on the fact that attackers don't know something needed to harm you.
  - Example: "If an attacker pointed their browser to <http://foo.com/passwords.txt>, they'd get our passwords. But nobody knows that file is there, so we are safe."
  - Example: "Our authentication database goes down for 2 minutes every night at 4am. During that time any user can log in without restrictions. But no one knows this, and the odds of a login at that time are miniscule."



# Secure authentication

---

- Force users to log in to your system before performing sensitive operations
- Use secure protocols (https, etc.) to prevent sniffing
  - Some sites use HTTPS only for login page, then switch back to regular HTTP for future page views. Is this bad?
- Force users to use strong passwords
  - not "password", or "abc", or same as their user name, etc.



# Principle of least privilege

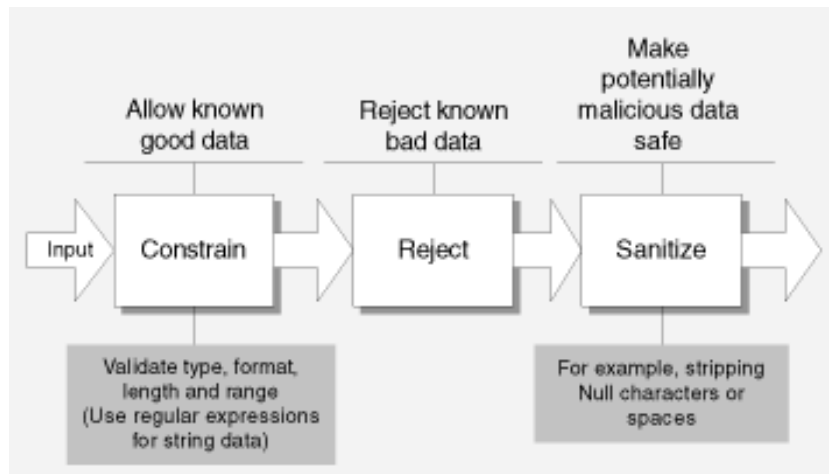
---

- **principle of least privilege:**  
Granting just enough authority to get the job done (no more!).
  - Examples:
    - A web server should only be given access to the set of HTML files that the web server is supposed to serve.
    - Code should not "run as root" or as a highly privileged user unless absolutely necessary.
  - Turn off unnecessary services on your web server
    - disable SSH, VNC, sendmail, etc.
    - close all ports except 80, and any others needed for web traffic



# Sanitizing inputs

- **sanitizing inputs:** Encoding and filtering untrusted user input before accepting it into a trusted system.
  - Ensure that accepted data is the right type, format, length...
  - Disallow entry of bad data into an HTML form.
  - Remove any SQL code from submitted user names.
  - HTML-encode input text that is to be displayed back to the user.



The screenshot shows a Mozilla Firefox browser window titled 'Untitled Document - Mozilla Firefox'. The address bar shows the URL 'http://localhost/filtering/filteringexample1.html'. The page content includes three questions with corresponding input fields: 'What is your name?' with 'Evil User', 'What is your favorite color?' with 'MIDNIGHT BLACK', and 'What is the airspeed of an unladen swallow?'. A JavaScript alert box is displayed, showing the code 'window.alert("GOTCHA!");' and '</script>'. At the bottom of the form is a 'Submit' button.

# Verifying that code is secure

---

- Before code is written:
  - considering security in the design process
- As code is being written:
  - code reviews
  - code security audits
  - pair programming
- After code has been written:
  - walkthroughs
  - system security audits
  - system/functional security testing
  - penetration tests



# Security audits

---

- **security audit:** A series of checks and questions to assess the security of your system.
  - can be done by an internal or external auditor
  - best if done as a process, not an individual event
- **penetration test:** Targeted white-hat attempt to compromise your system's security.
- **risk analysis:** Assessment of relative risks of what can go wrong when security is compromised.

# Security audit questions

---

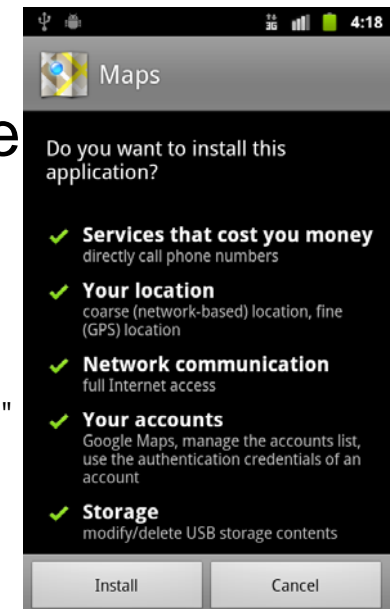
- Does your system require secure authentication with passwords?
- Are passwords difficult to crack?
- Are there access control lists (ACLs) in place on network devices?
- Are there audit logs to record who accesses data?
- Are the audit logs reviewed?
- Are your OS security settings up to accepted industry levels?
- Have all unnecessary applications and services been eliminated?
- Are all operating systems and applications patched to current levels?
- How is backup media stored? Who has access to it? Is it up-to-date?
- Is there a disaster recovery plan? Has it ever been rehearsed?
- Are there good cryptographic tools in place to govern data encryption?
- Have custom-built applications been written with security in mind?
- How have these custom applications been tested for security flaws?
- How are configuration and code changes documented at every level? How are these records reviewed and who conducts the review?

# Mobile app security

- On Android / iPhone, apps must be **signed** in market
  - manual approval reduces chance of rogue apps
- Apps must declare which **permissions** they need
  - e.g. use internet; write to local files; look at contacts; use Bluetooth; access GPS location; send SMS
  - user must manually give permission for actions
- Fine-grained access control in Manifest XML file
  - File/URL-specific permissions



```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.me.app.myapp" >
    <permission android:name="com.me.app.myapp.permission.DEADLY_ACTIVITY"
        android:label="@string/permlab_deadlyActivity"
        android:description="@string/permdesc_deadlyActivity"
        android:permissionGroup="android.permission-group.COST_MONEY"
        android:protectionLevel="dangerous" />
</manifest>
```



# Attack exercise

---

- We are taking CSE 144, which uses an online turnin system.
  - We want to hack it because we are evil.
- Our goals:
  - We want to cheat on Homework Assignment 7, Song.java. We want to find a way to submit a perfect working solution without doing any real work.
  - We got a low grade on a past assignment, so if possible, we want to set our past grades to be higher than they are now.
  - Our enemy is fellow classmate Felix Chu. We want to find out his personal information (password, email, student ID, grade, etc.).
  - We don't like the course instructor, Marty Stepp. We want to make the turnin page print an embarrassing message about him.



# Thank you!

