



CSE 403

Lecture C

Security Testing for Mobile and Web Apps

slides created by Marty Stepp
<http://www.cs.washington.edu/403/>

Outline

- Designing for security
- Web security
- Thinking like an attacker: finding vulnerabilities
- Android security risks



Designing for security

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 app saves its sensitive user data using SQLite which ends up as a file on the local file system."
 - 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.

Authentication Required

Enter username and password for "" at <https://pascal.cs.washington.edu>

User Name:

Password:

Use Password Manager to remember this password.

OK Cancel

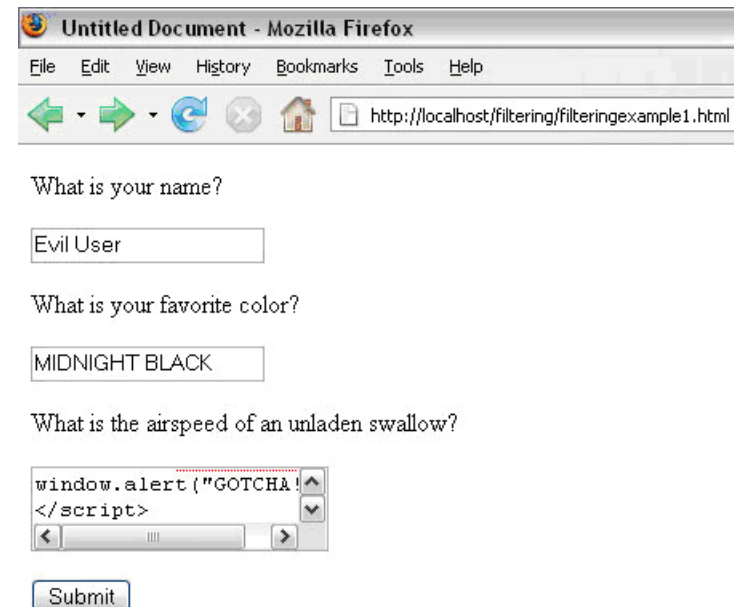
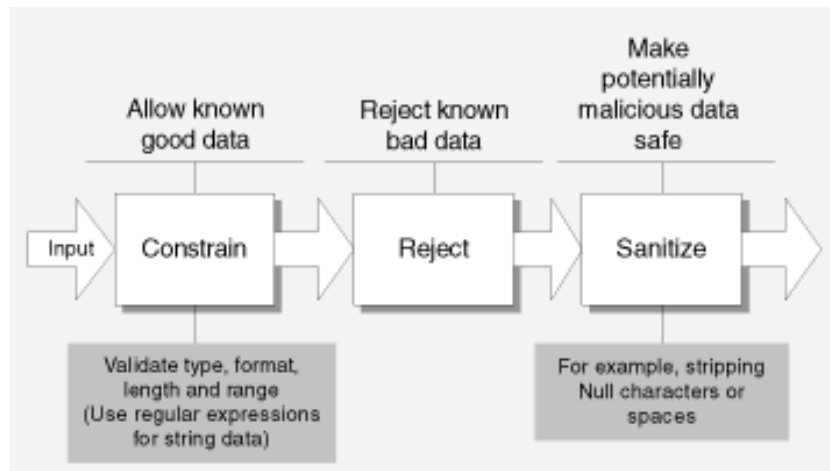
Principle of least privilege

- **principle of least privilege:**
Granting just enough authority to get the job done (no more!).
 - Examples:
 - Code should not "run as root" or as a highly privileged user unless absolutely necessary.
 - A web server should only be given access to the set of HTML files that the web server is supposed to serve.
 - Turn off unnecessary services on your 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 a graphical form.
 - Remove any SQL code from submitted user names.
 - Encode/sanitize input text that is displayed back to the user.



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?

Data classification

- **data classification table:** For each kind of data your app saves/uses, ask yourself:
 - Is this information personal or sensitive in nature?
 - What does my app do with this information?
 - Where and in what format is it saved?
 - Is it sent over the network?
 - (for all above) Does it need to be? Can I reduce my use?

Data Type	Personal?	Sensitive?	Create	Store	Send	Receive
Name	Yes	No	X	X	x	
E-mail Address	Yes	Yes	X	X	x	
Phone No.	Yes	Yes	X	X		
Address	Yes	Yes	X	X		

Data storage location

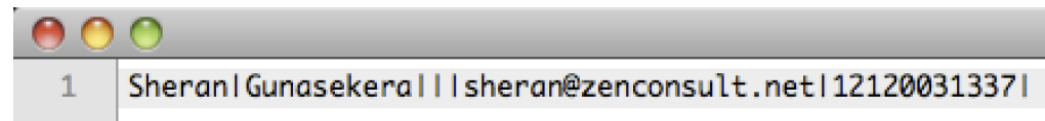
- Where is your app storing its data, and why? Is it secure?

Storage Method	Description	Data Privacy
Shared preferences	Allows you to store primitive data types (e.g., int, Boolean, float, long, and String) that will persist across the device session. Even if your application is not running, your data will persist until the device is restarted.	Can set four modes of privacy: MODE_PRIVATE, MODE_WORLD_READABLE, MODE_WORLD_WRITEABLE, and MODE_MULTI_PROCESS. Default mode is MODE_PRIVATE
Internal storage	Allows you to store your data in the device's internal memory. Generally, this data is not accessible by other applications or even the end user. This is a private data storage area. Data stored here will persist even after a device restarts. When the end user removes your application, Android will also delete your data.	Can set three modes of privacy: MODE_PRIVATE, MODE_WORLD_READABLE, and MODE_WORLD_WRITEABLE. Default mode is MODE_PRIVATE.
External storage	Data stored here is world-readable. The device user and other applications can read, modify, and delete this data. The external storage is associated with SD Cards or device internal storage (which is nonremovable).	Data is world readable by default.
SQLite databases	If you need to create a database for your application to take advantage of SQLite's searching and data management capabilities, use the SQLite database storage mechanism.	Databases that you create are accessible by any class within your application. Outside applications have no access to this database.
Network connection	You can store and retrieve data remotely through web services. You can read more	Based on your web service settings.

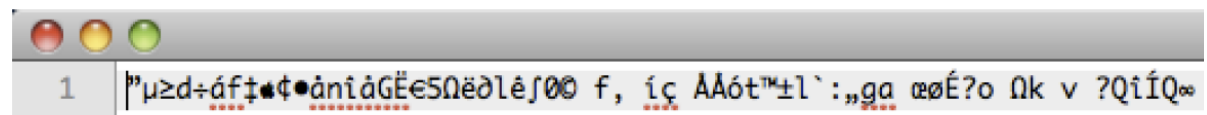
Encryption

- You can easily **encrypt** data in Android just before/after saving it to the device's SD card or local database.

```
private static byte[] encrypt(byte[] key, byte[] data) {  
    SecretKeySpec sKeySpec = new SecretKeySpec(key, "AES");  
    Cipher cipher;  
    byte[] ciphertext = null;  
    try {  
        cipher = Cipher.getInstance("AES");  
        cipher.init(Cipher.ENCRYPT_MODE, sKeySpec);  
        ciphertext = cipher.doFinal(data);  
    } catch (NoSuchAlgorithmException e) {  
        Log.e(TAG, "NoSuchAlgorithmException");  
    } catch (InvalidKeyException e) {  
        Log.e(TAG, "InvalidKeyException");  
    } catch (Exception e) {  
        Log.e(TAG, "Exception");  
    }  
    return ciphertext;  
}
```



```
1 Sheran|Gunasekera||sheran@zenconsult.net|121200313371
```



```
1 |μzd+áfııç•ániáGĒe5Ōēðlêj00 f, íç AAót™±l`:,„ga æøÉ?o Ōk v ?QiÍQ∞
```

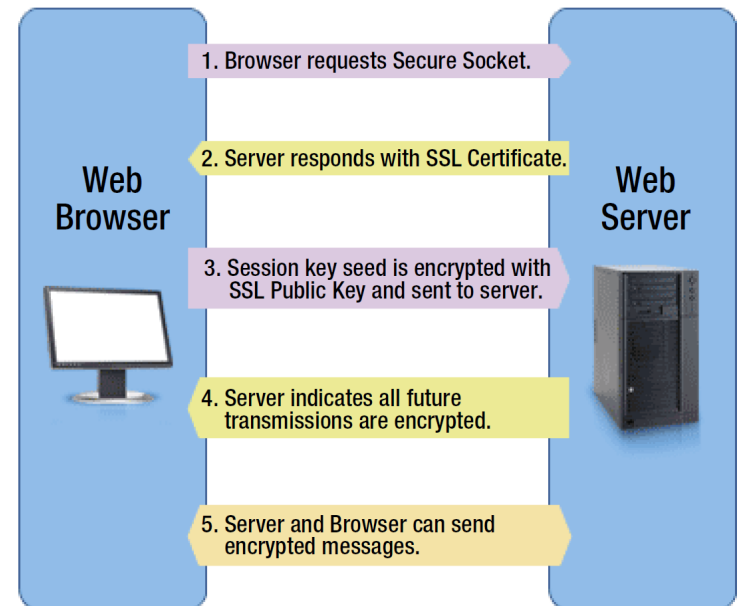
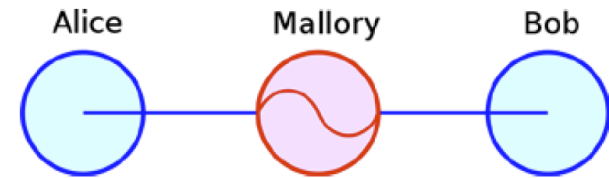
Mobile+web apps



- OWASP Top 10 issues for mobile apps that talk to web apps:
 - **Identify and protect** sensitive data on the mobile device.
 - Handle **password credentials** securely on the device.
 - Ensure that sensitive data is **protected in transit**.
 - Implement user **authentication** and session management correctly.
 - Keep the **back-end APIs** (services) and the platform (server) secure.
 - Perform data integration with **third party** services/apps securely.
 - Pay specific attention to the collection and storage of **consent** for the collection and use of the user's data.
 - Implement controls to prevent unauthorized access to **paid-for** resources (e.g., wallet, SMS, and phone calls).
 - Ensure secure **distribution**/provisioning of mobile applications.
 - Carefully check any **runtime interpretation** of code for errors.

Secure web (HTTPS)

- **man-in-the-middle** attack:
 - unauthorized third party can hear web traffic on its hops between client and server
- For security, all web traffic in your app should use HTTPS secure protocol.
 - built on Secure Socket Layer (SSL)





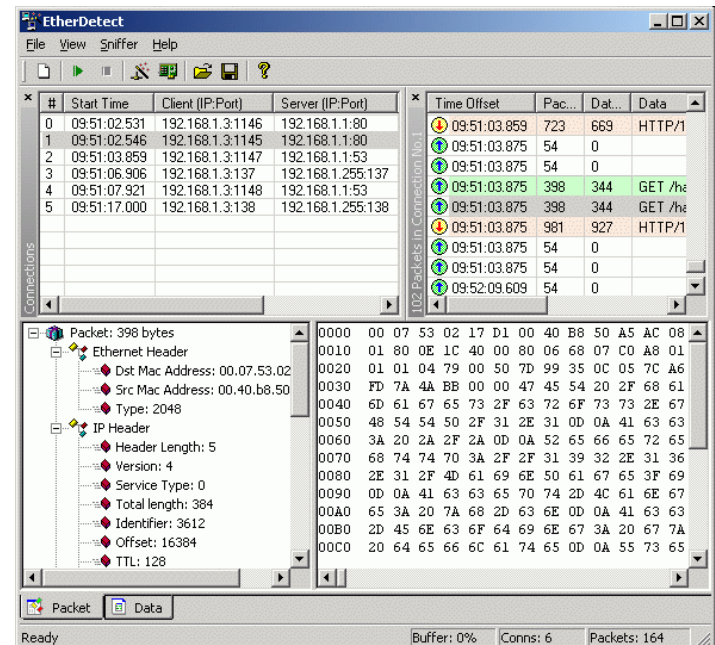
Web security

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/webscr/cmd_login.php

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

Thank you for using PayPal!

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).

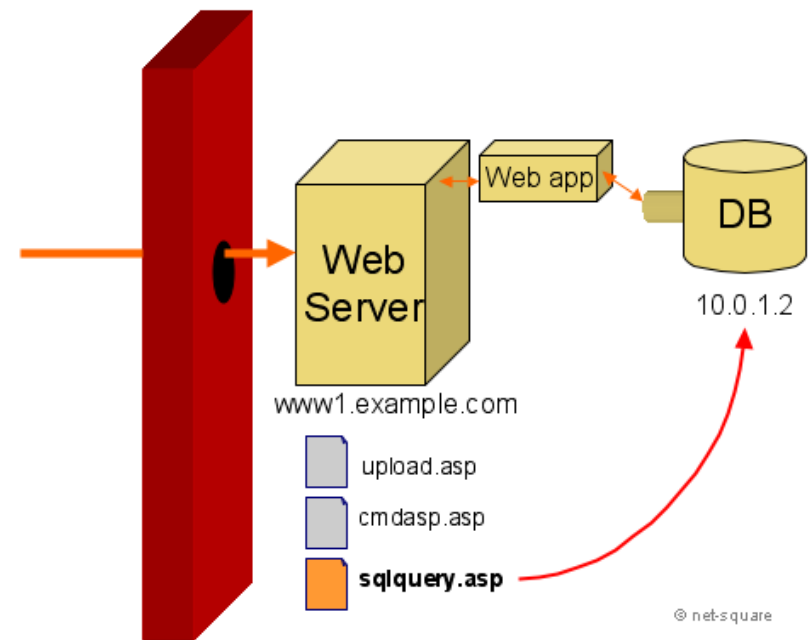
PayPal will never ask you to enter your password in an email.

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

Protect Your Password

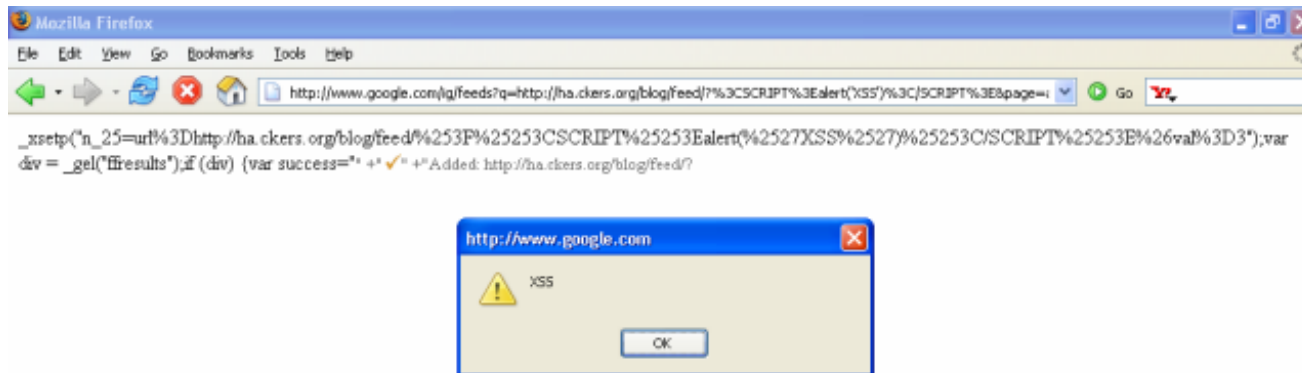
Privilege escalation

- **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';
```





Thinking like an attacker: finding vulnerabilities

Panning for gold

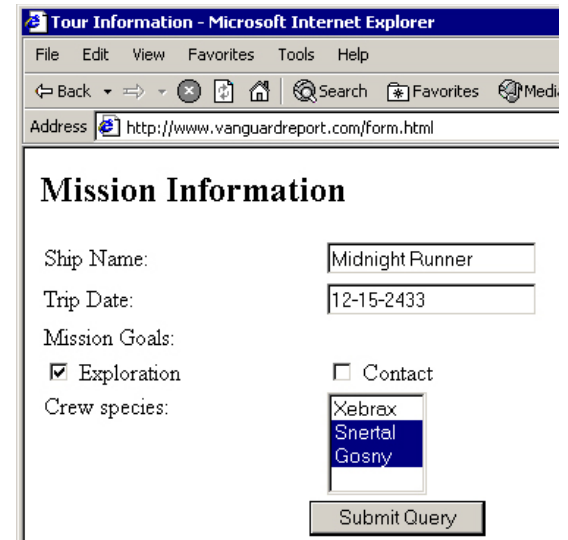
- **View Source**, and look for:
 - HTML comments
 - script code
 - other sensitive information in code:
IP/email addresses, SQL queries, hidden fields,...
- watch HTTP requests/responses
 - look for hidden pages, files, parameters to target
- error messages sent to your browser by app
 - 200: OK
 - 400: Invalid request
 - 403: Forbidden
 - 404: File not found
 - 500: Internal server error



```
<html xmlns="http://www.w3.org/1999/xhtml" >
  <head><meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
  <meta name="Generator" content="iWeb 4.0.3 (20081027.1625)" />
  <meta name="iWeb-Build" content="iWeb 4.0.3 (20081027.1625)" />
  <meta http-equiv="X-UA-Compatible" content="IE=EmulateIE7" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title>huan_ying.html</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_yingIE7.css" />
  <!--[if gte IE 8]><link rel="stylesheet" type="text/css" href="Media/IE8.css" /><![endif]>
  <script type="text/javascript" src="mmon.js"></script>
  <script type="text/javascript" src="pt">
  <script type="text/javascript">
  </script>
  </head>
  <body style="background: rgb(255, 255, 255); color: black; font-family: Arial, Helvetica, sans-serif; font-size: 12px; text-align: center;" onunload="onPageUnload();">
    <div style="text-align: center;">
      <div style="margin-bottom: 10px; margin-top: 0px; overflow: hidden; padding: 0px 0px 0px 0px; background: rgb(255, 255, 255); text-align: center;">
        <div style="margin-left: 0px; width: 100%; height: 0px;">
```

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...

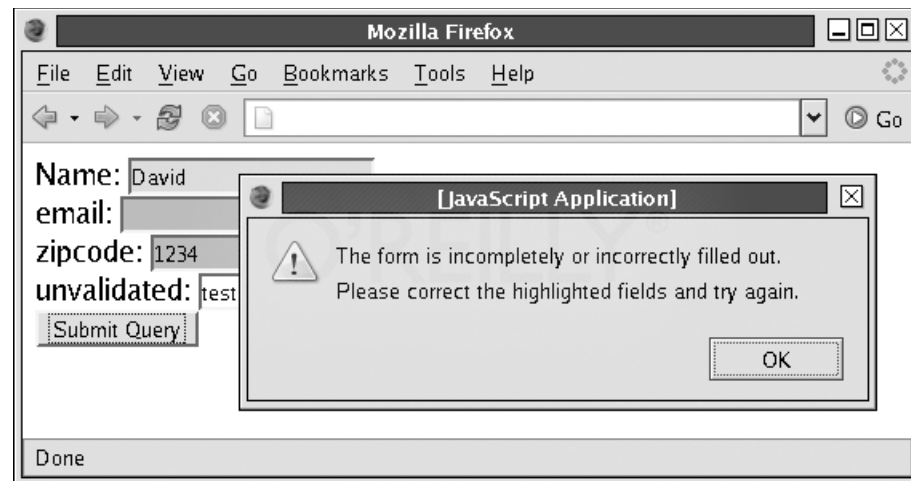


The screenshot shows a Microsoft Internet Explorer browser window titled "Tour Information - Microsoft Internet Explorer". The address bar displays "http://www.vanguardreport.com/form.html". The main content area contains a form titled "Mission Information" with the following fields:

- Ship Name:
- Trip Date:
- Mission Goals:
 - Exploration
 - Contact
- Crew species:
-

Form validation

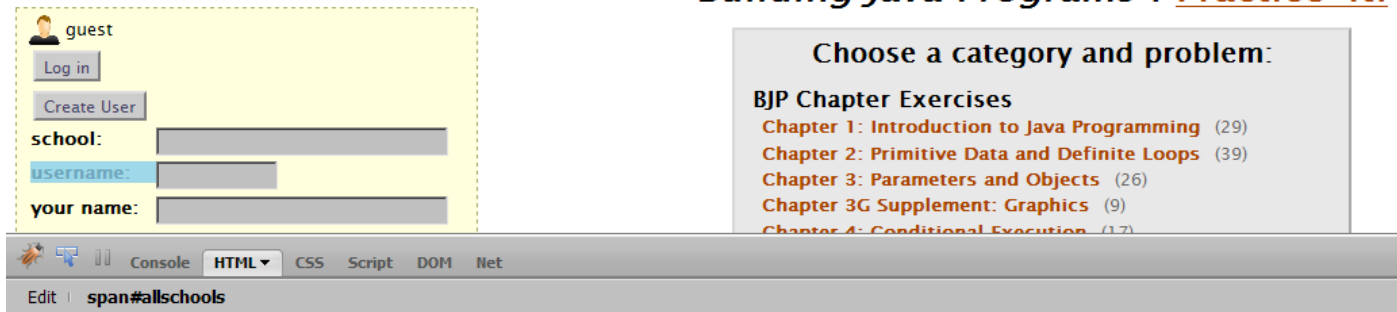
- **validation**: Examining form parameters to make sure they are acceptable before/as they are submitted.
 - nonempty, alphabetical, numeric, length, ...
 - **client-side**: HTML/JS checks values before request is sent.
 - **server-side**: JSP/Ruby/PHP/etc. checks values received.
- Some validation 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)

```
<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)
- ...



Android security risks

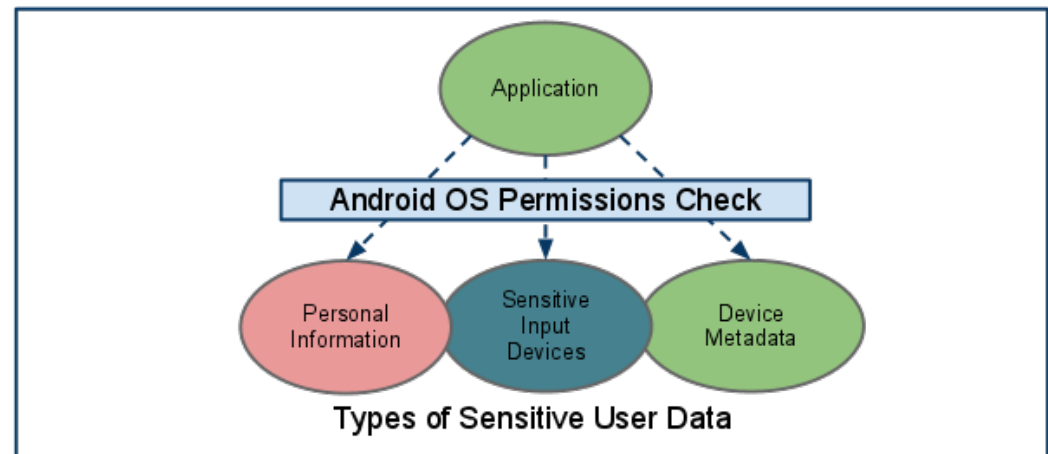
Android security risks

- What are some security risks you can think of that can affect an Android phone?
 - What are actions a malicious app could take?
- Examples:
 - uses a bug or security vulnerability to gain ungranted permissions
 - shows the user unsolicited messages (especially commercial)
 - resists (or attempts to resist) the user's effort to uninstall it
 - attempts to automatically spread itself to other devices
 - hides its files and/or processes
 - discloses the user's private information to a third party w/o consent
 - destroys the user's data (or the device itself) without w/o consent
 - impersonates the user (such as by sending email or buying things)
 - drains the phone's battery, data bytes/minutes, SMS/MMS remaining
 - otherwise degrades the user's experience with the device



Android OS security

- The Android operating system provides security:
 - Unix-based file/directory permission model
 - process memory isolation and memory protection
 - filesystem encryption
 - per-app access to hardware devices
 - per-app restrictions on memory/CPU usage, other resources
 - network/data connection
 - camera
 - location (GPS) data
 - bluetooth
 - SMS/MMS
 - ...
 - DRM framework

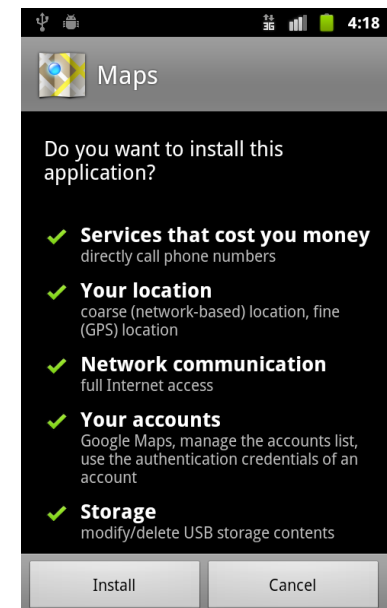


Mobile app permissions

- 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.android.app.myapp" >
  <uses-permission android:name="android.permission.RECEIVE_SMS" />
  ...
</manifest>
```



Signed apps/stores

- **signed apps:** Coded with a private developer key
 - On Android / iPhone, apps must be signed in market
 - manual approval reduces chance of rogue apps
 - any app bought in official App Store / Market is generally thought of as having being audited
 - Is this true for Apple store apps?
 - Is this true for Google Play Market apps?
 - App store users can **rate** the apps and **comment**
 - Do you feel that an app is more likely to be secure:
 - If it is from a publisher/company you already know?
 - If a friend of yours has it installed?
 - If it costs money?



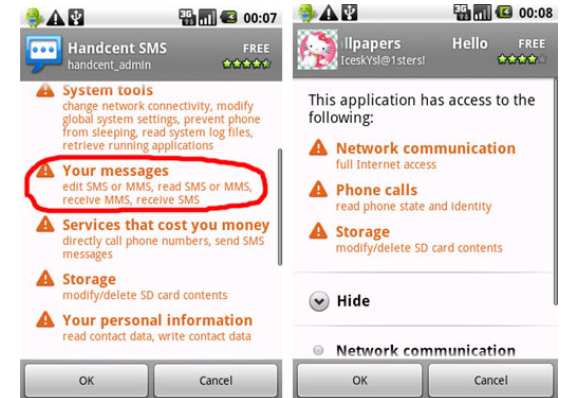
Google play



App Store

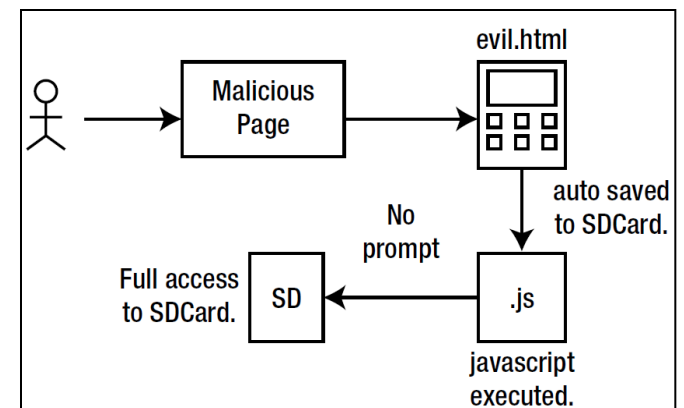
Problems with Android

- Apps can ask for too many permissions.
 - Users don't really understand permissions.
 - Users are overwhelmed and just click "Yes"
 - Now the app can do almost anything.
- Updates to an app can change its permissions.
 - example: recent Facebook app update
 - Users often click "Yes" if they already trust the app.
 - "privilege escalation"
- Spammy apps
 - resist attempts to uninstall
 - show ads that are like system/OS UI
 - disclose or damage the user's personal information data
 - impersonates the user



Example attack

- Android 2.2 / 2.3 had vulnerabilities.
 - Browser could download a HTML page.
 - The page contains JS code.
 - The JS code can self-execute later in a "local" context.
 - This has higher permissions and can modify the local file system.



- App ABC stores sensitive data on the local file system.
 - The data is financially important.
 - It is saved as a file in plain-text.
 - The above malicious browser JS code can read and access it.