Security Testing: Principles and Practices
SFWR ENG 3S03: Software Testing

Alicia Marinache

Department of Computing and Software, McMaster University
Canada L8S 4L7, Hamilton, Ontario

Acknowledgments: Material adapted from Dr.R.Paige, Dr. R. Khedri, [WNZD07, Chaper 2]

- Security testing ensures the software is protected against security threats
- It verifies the software's ability to defend against attacks
- Focuses on vulnerabilities, unauthorized access, data breaches, etc.
- Two major categories
  - design vulnerabilities
  - implementation vulnerabilities

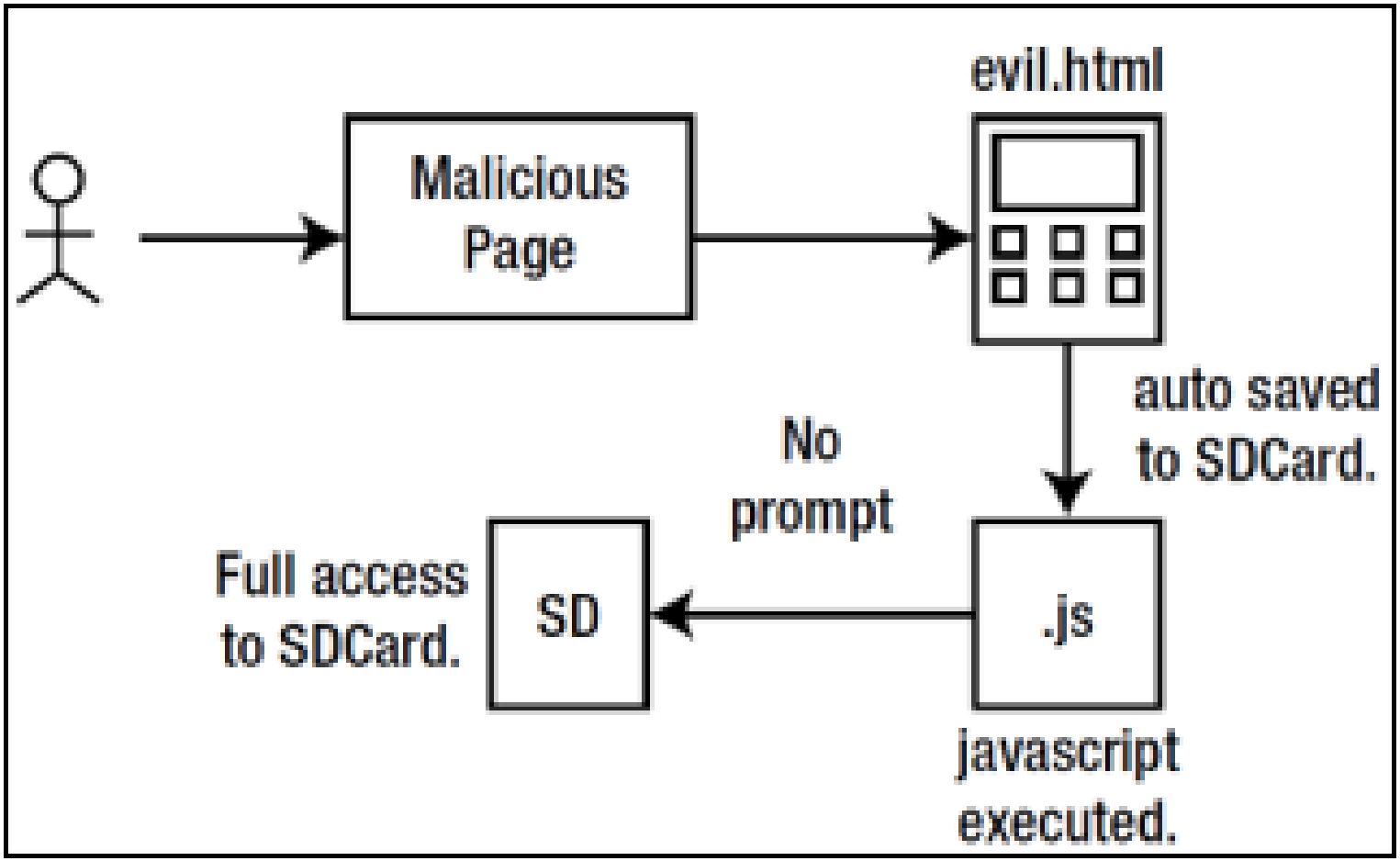
Objectives of Security Testing

- Identify vulnerabilities and weaknesses in software
- Protect against malicious attacks and data breaches
- Ensure compliance with security standards (e.g., OWASP, GDPR)
- Verify the effectiveness of security controls and features

Focus: Mobile and Web Applications Security

- Many attack vectors
- Many devices, not always up-to-date
- Powerful capabilities on the device/web client
- Significant integration with cloud/servers

- Assume
  - Environment, browser, network
  - Potential attackers
  - Users are a menace to themselves (and their pets)
- Countermeasures
  - OS, browser, comm protocols protection
  - Resource & network monitoring
  - User awareness (e.g., encrypted folders, additional apps, turn on/off options)



Key Concepts in Security Testing

- **Integrity**: Ensures data is accurate, complete, and protected from unauthorized changes
- **Availability**: Ensures the system is available for use when required
- **Authentication**: Verifies the identity of users and devices
- **Authorization**: Defines what authenticated users are allowed to do
- **Confidentiality**: Ensures sensitive information is only accessible by authorized users

Security through obscurity: Relying on the fact that attackers don't know something needed to harm you

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

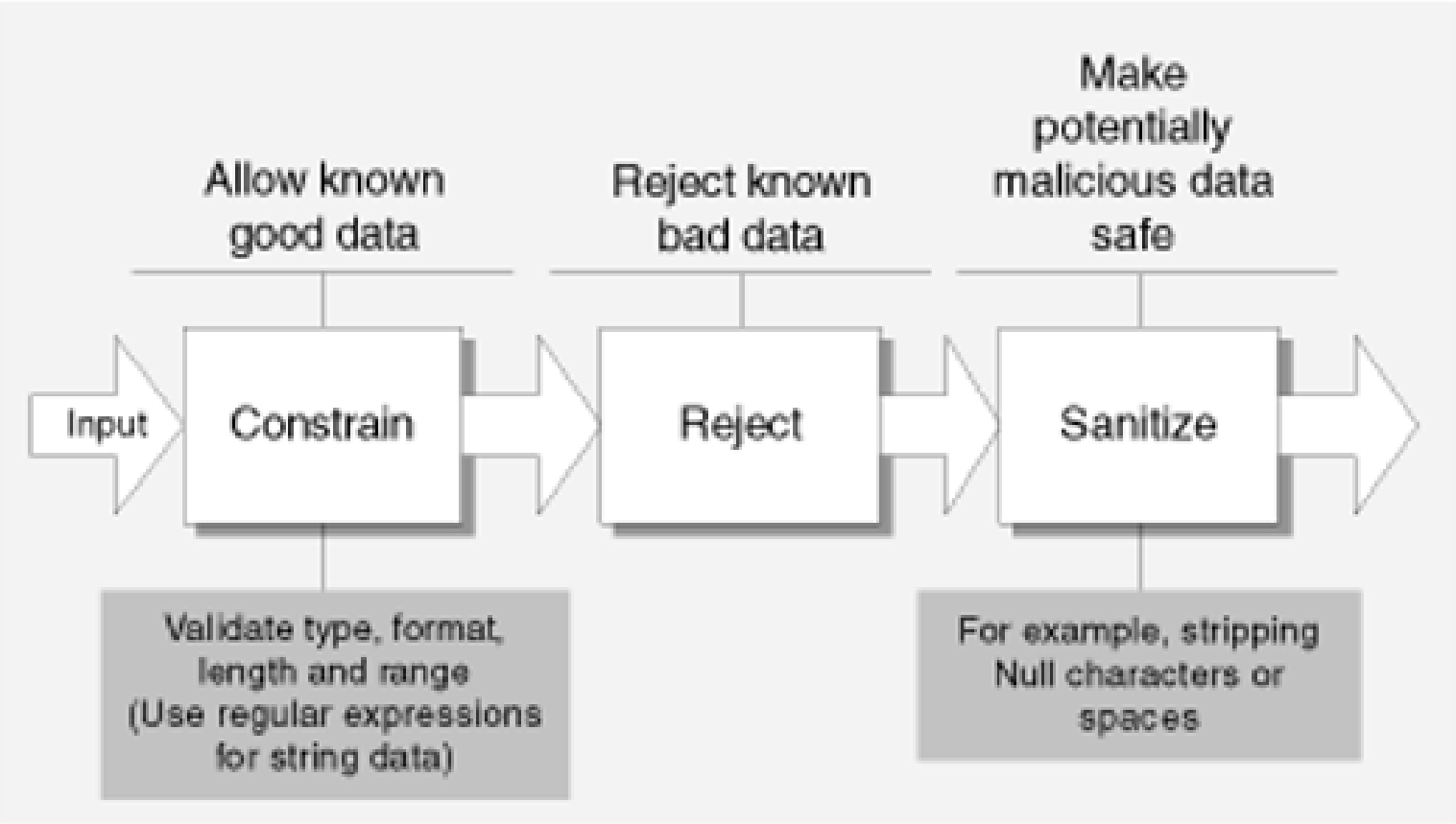
- Force users to log in before performing sensitive operations
- Use secure protocols
- Force users to use strong passwords
- Design and test for: password quality rules, default logins, password recovery, captcha, logout functionality, password change, security question/answer, etc.

Secure Authorization: Design Aspects

- Principle of least privilege
- Separation of privileges
- Design and test for: code running at higher privilege than absolutely necessary, unnecessary access to files and services, same user cannot initiate and approve sensitive actions, etc.

Data Integrity: Design Aspects

- Input validation: encoding and filtering untrusted user input before accepting it into a trusted system
  - Ensure that accepted data is the
  - Disallow entry of bad data into a form
  - Remove any SQL code from submitted inputs
- Centralize input validation
- Validate at component boundaries
- Design and test for: data is the right type, format, length (buffer overflow), form inputs, SQL code in submitted inputs (see SQL Injection), data flow between components (validate!), cookies, whitelist only allowed characters



- 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

Security Testing Lifecycle

- **Planning**: Define security objectives, scope, and resources + Audits
- **Design**: Identify security requirements and test scenarios + Audits
- **Execution**: Perform testing (e.g., penetration testing, vulnerability scanning) + Audits
- **Reporting**: Document vulnerabilities, severity levels, and recommendations + Audits
- **Closure**: Follow-up testing after remediation + Audits

Security Audits

- Before code is written
  - Consider security in the planning/design process
- (During) 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 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

- 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?

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

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_WRITABLE`, 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_WRITABLE`. 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.

Data Encryption

```
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|12120031337|
```

```
1  ”µ≥d÷áf‡•¢•ánìàGË€5Ωëðlêſ0© f, íç ÅÅót™±l`:„ga œøÉ?o Ωk v ?QîÍQ∞
```

OWASP (The Open Web Application Security Project): top 10 issues for mobile/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.

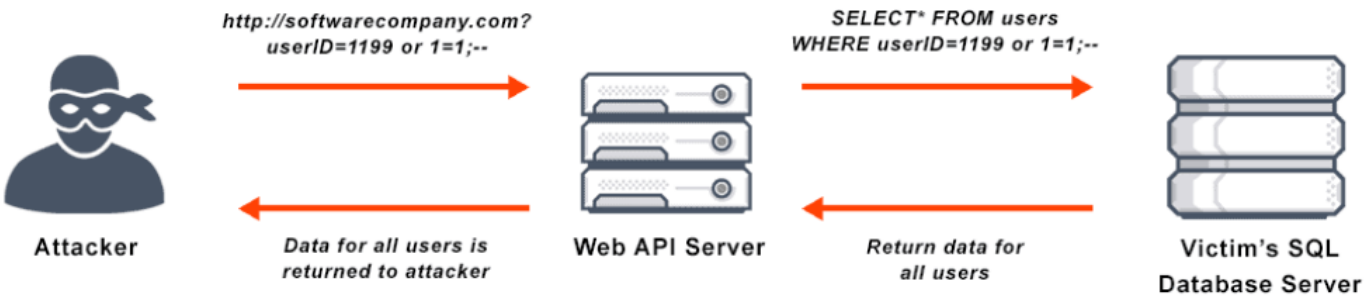
Mobile Web Security Challenges

- Mobile devices are more vulnerable due to **limited security resources** and **dynamic environments**
- Web browsers and mobile apps are often attacked due to **unsanitized inputs**, weak **authentication** systems
- Example: Attacks like **MITM**, **SQL Injection**, and **CSRF** are common

Common Security Threats in Web/Mobile Applications

- **SQL Injection**: Inserting malicious SQL queries into input fields
- **Cross-Site Scripting (XSS)**: Injecting malicious scripts into web pages
- **Cross-Site Request Forgery (CSRF)**: Forcing a user to execute unwanted actions on a site
- **Broken Authentication**: Exploiting weak authentication mechanisms
- **Sensitive Data Exposure**: Leaking sensitive information through insecure storage

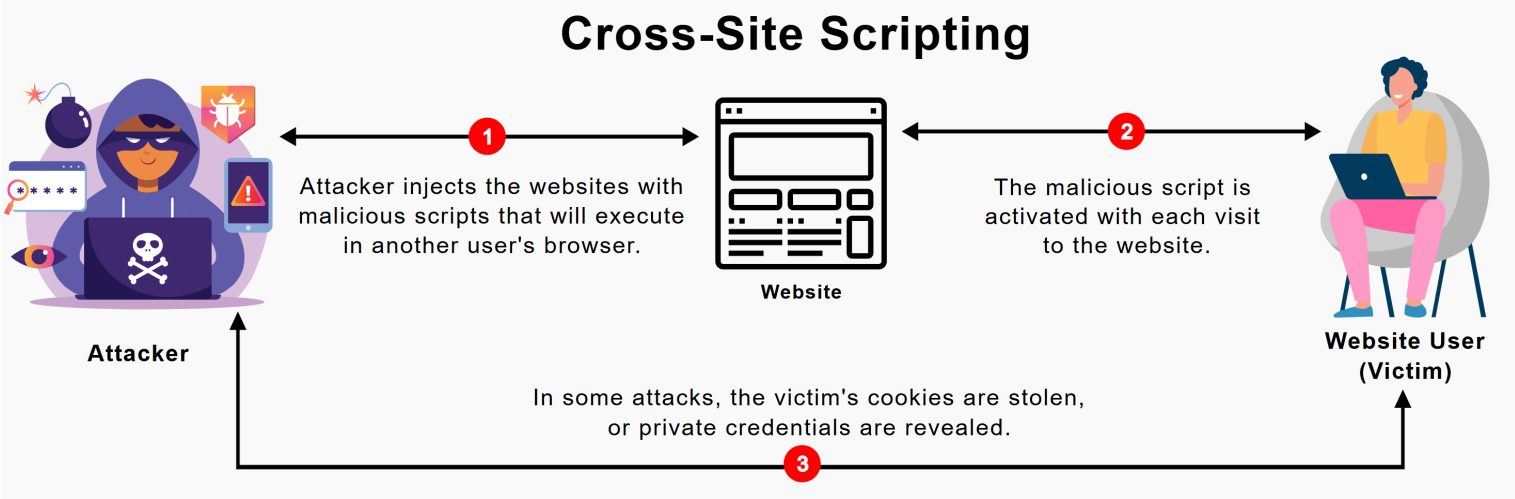
Example: SQL Injection in Web Applications



Source: `https://www.vmware.com/`

- Mitigation: use parameterized queries or prepared statements

Example: Cross-Site Scripting (XSS)



Source: `https://websitesecuritystore.com/`

- Mitigation: input sanitization and content security policies (CSP)

Security Testing in Mobile Applications

- Mobile apps face unique risks such as data leaks, improper session handling, and insecure storage
- Testing techniques
  - Reverse engineering: Decompiling or disassembling the code, to identify potential security weaknesses
  - Malware injection: Introducing malicious code into a system or application to test its defenses
  - Network sniffing: Monitoring network traffic for insecure data transmission
  - Session Hijacking: Stealing session tokens to gain unauthorized access

Tools for Security Testing

- OWASP ZAP: Open-source security testing tool for finding vulnerabilities in web applications
- Burp Suite: Widely used for **web vulnerability scanning** and **penetration testing**
- Wireshark: Network protocol analyzer for monitoring network traffic
- Frida: Dynamic instrumentation toolkit for **reverse engineering** and **mobile security testing**
- Nikto: Web scanner for identifying vulnerabilities like outdated software versions

Secure Development Lifecycle (SDLC)

- Good security design uses many overlapping techniques
- Security must be integrated throughout the entire software development lifecycle
- Steps include threat modeling, secure coding, and security testing
- Shift-left testing integrates security early in development to identify issues proactively

- Every programming language has idiosyncrasies that can lead to security flaws
- The programmer must avoid using some elements of a programming language or its programming environment to avoid creating implementation flaws
- Other language elements can be used safely as soon as the security implications of their usage are understood

References I

Chris Wysopal, Lucas Nelson, Dino Dai Zovi, and Elfriede Dustin, *The art of software security testing*, Addison Weslay, 2007.